CP-818A/U COMPUTER PROGRAMMER/ANALYST INSTRUCTION MANUAL

VOLUME I

INTRODUCTION TO COMPUTER PROGRAMMING

AUGUST 1969

TABLE OF CONTENTS

PREFACE

In the pages to follow we will discuss the evolution of the computer, its components, and methods involved in problem solving with a computer. This will also entail some exposure to numbering systems and the logic of Boolean Algebra. The intent of this manual is to provide the basics necessary for entry into computer programming instruction.

CHAPTER I

HISTORY OF DATA PROCESSING

THE THINKING MACHINE?

One of the hidden fears or phobias about computers current among the general population is the concept of the thinking machine. The question then is, "Do computers think?". This question in turn begs another question, "What is meant by the expression - thinking?" Webster quite loosely defines the word "think" as follows: to exercise the powers of judgement; to reflect for the purpose of reaching a conclusion. Using these definitions as a yardstick, we could say computers can be programmed to think. Thinking, however, is a word we use to mask many more complex operations by the brain, many of which we are just discovering. The president of a large aircraft company discussed this particular subject at great length when asked about the future of manned aircraft in the light of contemporary development in computer technology. His answer was, "Until someone builds for $100 or less with unskilled labor, a computer no larger than a grapefruit, requiring only a tenth of a volt of electricity, yet capable of digesting and transmitting incoming data in a fraction of a second and storing 10,000 times as much data as today's largest computers, the pilots of today have nothing to worry about."

MAN AND NUMBERS

Man's earliest use of digits as counters was probably a crude attempt to ascertain the continued attendance of his friends following an onslaught by wandering carnivores. When he passed into the herdsman and agriculturist stage of development, he probably used stones and knotted strings to keep track of his herds or individual animals. Strings of stones or beads were early means of counting. In many religions, beads are still used as prayer counters.

As commerce increased and life became more sophisticated, devices similar to the Abacus came into being as aids to computing. In this type of operation, the idea of counting is refined by using sliding beads on strands rather than strings. The abacus or soroban is a bi-quinary device (like two five-fingered hands), using five beads on one side of a wooden divider and two on the other side to count groups of fives. Simple counting is not the only way to compute. One can measure, also. Study of prehistory shows that early man observed the sun and moon and developed crude means of measuring the transit of heavenly bodies and passage of time. The arrangement of huge stones at Stonehenge in England, was started by unknown peoples in dark prehistory. It successfully told ancient priests when Midsummer's Day, eclipses and other key events of religious significance were about to occur. The basic idea behind sundials, modern clocks, slide rules,

thermometers, and automobile speedometers is the measurement of some secondary quantity which imitates or simulates the thing we wish to know. Thus, distance around a sundial simulates time, and the height of mercury in a thermometer is the analog of temperature.

Two types of computers, the analog and the digital, have evolved from the simple concepts of measuring and counting. The type of machine which most readily comes to mind when the term "computer" is used is the "digital computer" and this is the device we shall be discussing in this text. We should keep in mind the existence of many score thousand small analog computing devices called thermostats, planimeters, bathroom scales, and other articles that "compute" one thing by measuring another.

THE ANALYTICAL ENGINE

Some times history seems to go awry and events happen before they logically should. For instance, Leonardo Da Vinci sketched helicopters and war machines long before they could be built. Charles Babbage was a similar type. He was born in Devonshire, England, the day after Christmas, 1792. This banker's son conceived all the important features of modern digital computers; that is, the arithmetic unit, the memory, input and output schemes, and the most inventive concept of all, the idea of automatically sequencing instructions, by which a computer goes from one step to the next without human help.

The England of Babbage's day was a chaos of upheaval and change. Mathematics during the century following Sir Issac Newton's death had all but stagnated. In the period following the Napolenic Wars, those who were literate could not figure sums accurately. Financial accounts were snarled, logarithm tables were full of errors and actuary data for use in insurance were hopelessly jumbled. Babbage was bent on correcting all of this by using computing machines instead of people.

Babbage conceived his "difference engine" in 1820 and completed it in 1822. It was specially designed to compute polynomials, like $X^2 + 2X + 63$, for the preparation of mathematical tables. It was essentially a collection of gears and levers, similar to today's desk calculators, accurate to six places. With this success, Babbage tried to construct a better "difference engine," accurate to twenty places. He even talked the British government into contributing 17,000 pounds (an enormous sum in those days) to the project -- probably because of the military value of a device to prepare good ballistic tables. The project however, quickly became mired in manufacturing problems. The metal-working industry in the early 1800's could make smoothbore cannons and good plowshaves, but it wasn't competent for the precision gears and linkages described in Babbage's marvelously detailed drawings. So the project died, but not before he had trained a few machinists to make metal parts with more precision and detail than the world had ever seen.

Charles Babbage called his new dream the "Analytical Engine." It was designed to do all kinds of computations with the flexibility of a modern electronic computer. A vast assemblage of cogs, levers, and gears would be run by steam power. The Analytical Engine's memory was to consist of banks of wheels engraved with the ten digits. One thousand 50-digit numbers would be available upon demand by the "mill," where the arithmetic was done. Answers were to be automatically printed out just as computers do today. And, even more prophetic, the Analytic Engine was to control itself internally by punched cards.

The punched card concept was developed from the concept employed by Frenchman, Joseph Jacquard. Jacquard invented a mechanized loom system with punched cards controlling the pattern-weaving apparatus.

Babbage died in London in 1871, leaving a legacy we have yet to fully understand and measure. He wrote: "If, unwarned by my example, any man shall succeed in constructing an engine embodying in itself the whole of the executive department of mathematical analysis -- I have no fear of leaving my reputation in his charge, for he alone will be able to fully appreciate the nature of my efforts and the value of their results."

THE EVOLUTION OF THE ANALYTICAL ENGINE

One fundamental concept of modern computers that Babbage seems not to have had is that of the stored program. His instruction cards were apparently not alterable by the machine in the course of computations. The instructions were recorded only in the cards and were not stored in a storage device comparable to the device used for data. Therefore, instructions could not be altered by the arithmetic unit. Moreover, the machine had no ability to jump arbitrarily from one instruction to a remote instruction in the program.

For many years advances in computing devices were limited to the development of adding machines, desk calculators, punched card accounting machines and the like. All of these have been useful and successful commercially but did not approach the computing power that could have been achieved through the use of Babbage's ideas.

THE MARK I COMPUTER

After Babbage died, no major advance in automatic computation was made until 1937, when Professor Howard Aiken of Harvard University became interested in combining some established principles with the punched cards pioneered by Hollerith and Powers to build an automatic calculating device. In May, 1944, with the cooperation of IBM, an automatic-sequence-controlled calculator named the Harvard Mark I was built and formally presented to Harvard University.

Aiken's machine was built on the concept of using information from punched cards as input, making decimal calculation through electro-mechanical devices, and producing the results on punched cards again. The sequence of calculations was controlled through a wide-punched paper tape.

The machine was adapted to solve various kinds of problems for engineers, physicists, and mathematicians, and was the first machine to do long series of arithmetic and logical problems. After the Mark I, Professor Aiken also constructed three more models.

The Mark I computer is considered to be the first successful general-purpose digital computer.

THE ENIAC

In the early 1940s, Dr. John W. Mauchly of the University of Pennsylvania became aware of the need for a high-speed electronic device able to do great quantities of statistical calculations for weather data. During WW II, a contract for the project was made between the University of Pennsylvania and the U. S. Government.

In 1945, Dr. Mauchly and J. P. Eckert used the facilities at the Moore School of Electrical Engineering to design and build the Electronic Numerical Integrator and Calculator (ENIAC). ENIAC was completely electronic in that it had no moving parts other than input/output gear. It was installed at the Aberdeen Proving Grounds in Maryland and was used until 1956, when it was removed to be placed in the Smithsonian Institute.

ENIAC was a large machine, containing 18,000 vacuum tubes. It had a small memory of 20 accumulators for storing data. Each accumulator was capable of carrying 10 digits. The accumulators consisted of vacuum tubes, tied together in packages of two, so that two tubes would represent one binary digit (bit) in the computer storage. The machine was externally programmed, but had internal capacities of a multiplier, divider, and three function tables. Input and output were made through punched cards. In the mid-1940s, the best available electromechanical equipment could perform a maximum of one multiplication per second. By contrast, ENIAC could process 300 multiplications per second.

Although many tube-type computers are still operating satisfactorily, the advent of the transistor (commonly referred to as "solid state") has greatly improved the efficiency of computer operations. A transistor is much smaller than a vacuum tube. Solid state computers are much more compact than tube type computers; they require less rigid air-conditioning, have a greater life expectancy, and are less susceptible to failure.

Computer manufacturers are constantly trying to develop computers with a longer life and of a smaller size capable of storing

more data and operating at greater speeds.

In recent years, the trend has been toward producing medium-scale computer systems. The larger, more powerful computer systems have been used primarily for scientific and engineering purposes. Current and future trends may very likely move toward the production of small scale computers, especially since electronic data processing has made a powerful appeal to the small business man.

CHAPTER II

COMPUTER PROGRAMMING NUMBER SYSTEMS

I. General Features of All Number Systems

II. Counting

III. Arithmetic Operations

- A. Addition
  1. Binary
  2. Octal
- B. Subtraction
  1. Binary
  2. Octal
- C. Multiplication
- D. Division

IV. Conversion of Whole Numbers from One System to Another

- A. From decimal to octal
- B. From decimal to binary
- C. From octal to binary
- D. From binary to octal
- E. From binary to decimal
- F. From octal to decimal

V. Conversion of Fractions from One System to Another

- A. From decimal to octal
- B. From decimal to binary
- C. From octal to binary
- D. From binary to octal
- E. From octal to decimal
- F. From binary to decimal

VI. Rounding Your Answers

VII. Answers

NUMBER SYSTEMS

I. General Features of All Number Systems.

There is much speculation as to the manner in which counting first took place. Undoubtedly it started with the keeping track of various objects be they sheep, bags of grain, or wives. Possibly a sheepherder (better known as a shepherd) had the problem of keeping track of the flock and did not know how to count too well. A solution to his problem would be to take a handful of pebbles and place one in his right hand pocket as he counted each sheep leaving. Then at the end of the day to check up on the flock he would transfer a pebble from his right hand to his left hand pocket as the sheep went by. If there were no pebbles left over at the end of the day, his herd was taken care of. To keep track during the day and avoid running around in the middle of the night trying to find lost sheep the herder soon started counting the sheep every few hours. For this purpose it was much nicer to use sticks laid out in a row. He soon grouped the sticks into fives to correspond to the fingers on one hand and later may have placed the fifth stick across the rest. This method of representing quantities is called the tally method of counting. We can see that it has the disadvantage of being very unwieldy for larger quantities. To write the number representing one million items in this system would take a fortnight or more if we made one mark a second.

The other extreme would be to have a separate sumbol for each quantity. If we wanted to represent a million units we would put down a symbol which would take less time than it would take us to write 1,000,000. The disadvantage would be the difficulty of remembering all the symbols and being able to recall them with some facility.

Just as we count on our fingers today, so someone in early times started counting on their fingers and probably developed a symbol for one hand which equated to a quantity of five. Both the Egyptians and Romans seem to have come up with symbols for ten, hundred, etc. The Egyptian maximum quantity symbol was for one million; where the Roman maximum quantity symbol was for one thousand.

II. Counting.

We all count and do other arithmetic operations with facility in the decimal system because of long years of association with this method. These operations depend on certain rules that also apply to other number systems. Since it may be difficult to translate the process of counting in the decimal system into counting in another system, here is a good place to review the basic rules for counting.

1. Starting from a position at the right, the number can have any number of positions to the left.

2. In each position we must have the mark 0, or one of the other admissible marks (symbols).

3. To obtain the number one greater than a given number, raise the extreme right-hand mark to the next higher admissible mark. If this is already the highest admissible mark, go left to the first position where the mark is not the highest one possible, raise this mark to the next mark, and set everything at the right to 0 (the lowest mark). What these rules say essentially is that we can have any number of digits in our number, there are certain symbols (such as 0, 1, 2, 3, etc.), and that when we reach the highest symbol of our system we go to the left and raise the symbol in the next column to the next higher symbol and then set all symbols to the right of the symbol which we raised to the lowest symbol, zero.

If you don't understand the preceding, don't be overly concerned. You understand the principles but have probably never seen them expressed in writing.

If you have the decimal number 666, all of the sixes do not represent the same quantity. The right-most 6 represents six units, the next 6 - sixty units, and the next 6 - six hundred units. For this reason, we have a sumbol-positional system. The symbol we use and the position it occupies determine the quantity that it represents. In the decimal system we have ten symbols (0,1,2,3,4,5,6,7,8,9).

Let us look at the number 4546:

This is equal to $4000 + 500 + 40 + 6$
This is equal to $4 \cdot 1000 + 5 \cdot 100 + 4 \cdot 10 + 6 \cdot 1$
This is equal to $4 \cdot 10^3 + 5 \cdot 10^2 + 4 \cdot 10^1 + 6 \cdot 1$

(Dot means multiply, e.g., $6 \cdot 5$ means 6 times 5)

The number 79.42 equals $70 + 9 + 4/10 + 2/100$
which equals $7 \cdot 10 + 9 \cdot 1 + 4 \cdot 1/10 + 2 \cdot 1/100$
which equals $7 \cdot 10^1 + 9 \cdot 10^0 + 4 \cdot 10^{-1} + 2 \cdot 10^{-2}$

(Remember: Any number raised to the zero power is one)

From the above illustrations we can see a pattern in which the power of ten decreases by one as we go from left to right. (NOTE the fact that starting from the decimal point and going left, we have 10^0, 10^1, etc. and going right we have 10^{-1}, 10^{-2}, etc. The same is true for all number systems - so in binary we would start at binal point (same

as decimal point) and going to the left we would have 2^0, 2^1, etc. and going right we would have 2^{-1}, 2^{-2}, etc. In the octal system we would start at octal point and going to left 8^0, 8^1, 8^2, etc., and going right 8^{-1}, 8^{-2}, 8^{-3}, etc.)

We can make the following equivalence:

Number 79.42 equals: $7\cdot10^1 + 9\cdot10^0 + 4\cdot10^{-1} + 2\cdot10^{-2}$

So: $N = a_1\cdot10^1 + a_0\cdot10^0 + a_{-1}\cdot10^{-1} + a_{-2}\cdot10^{-2}$

If we had 2054.617 then:

$$2054.617 = 2\cdot10^3 + 0\cdot10^2 + 5\cdot10^1 + 4\cdot10^0 + 6\cdot10^{-1} + 1\cdot10^{-2} + 7\cdot10^{-3}$$

$$N = a_3\cdot10^3 + a_2\cdot10^2 + a_1\cdot10^1 + a_0\cdot10^0 + a_{-1}\cdot10^{-1} + a_{-2}\cdot10^{-2} + a_{-3}\cdot10^{-3}$$

$$N = a_3\cdot r^3 + a_2\cdot r^2 + a_1\cdot r^1 + a_0\cdot r^0 + a_{-1}\cdot r^{-1} + a_{-2}\cdot r^{-2} + a_{-3}\cdot r^{-3}$$

$$N = a_m\cdot r^m + a_{m-1}\cdot r^{m-1} + \ldots a_{-m+1}\cdot r^{-m+1} + a_{-m}\cdot r^{-m}$$

We could generalize this in the following way:

$$N = a_m\cdot r^m + a_{m-1}\cdot r^{m-1} + \ldots a_1\cdot r^1 + a_0\cdot r^0 + a_{-1}\cdot r^{-1} \;.\;.\; + a_{-m}\cdot r^{-m}$$

a_{m-1} etc. stands for constants

r stands for radix or base (total number of admissible symbols in the system)

m stands for position of the admissible symbol.

The above generalization is true for all number systems of the symbol positional type.

Unfortunately, it is rather difficult to get ten different states of an electric current or voltage, but it is very, very simple to get a two-state condition: either it is off or on, it is either positive or negative, it is either positive or zero, etc. We can set up a very simple situation by having the circuit go through a light bulb. Then when current is flowing we will see the light on and when it is not flowing the light will be off. In addition, we can take advantage of

binary logic and use "and" and "or" gates. We may represent a light being on by the symbol "1" and off by the symbol "0". Suppose we had the following: 1010.

Then we would have: $1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$

This equals: $1 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 0 \cdot 1$

$8 + 0 + 2 + 0 = 10$

So: $1010_2 = 10_{10}$

The binary system has some of the shortcomings of the tally system. Namely - it takes many symbols to represent large numbers. If we had a string of 9 lights on a computer it would be rather difficult to read them with any facility; in fact, research shows that we cannot effectively see more than six separate objects at the same time. If we group our lights into groups of three we find that we simplify the reading of the 9 lights.

Instead of seeing, for example: 1 1 0 0 1 0 1 1 1

we see: 110 010 111

Let us see what quantities we can represent with 3 binary digits or 3 bits:

000 = 0
001 = 1
010 = 2
011 = 3
100 = 4
101 = 5
110 = 6
111 = 7

Since this gives us a system which represents 8 different quantities (0-7) we have an octal system. Thus we count 0,1,2,3,4,5,6,7 (now we have reached our highest symbol so we go the the left and raise this symbol, which is a "0", even though it is not written, to a "1" and set the symbols to right, we only have one in this case, to the lowest symbol or "0")

Going back to our	110	010	111
We can write this in octal form as	6	2	7

$$110010111_2 = 1\cdot2^8 + 1\cdot2^7 + 0\cdot2^6 + 0\cdot2^5 + 1\cdot2^4 + 0\cdot2^3 + 1\cdot2^2 + 1\cdot2^1 + 1\cdot2^0$$

$$= 1\cdot256 + 1\cdot128 + 0\cdot64 + 0\cdot32 + 1\cdot16 + 0\cdot8 + 1\cdot4 + 1\cdot2 + 1\cdot1$$

$$= 256 + 128 + 0 + 0 + 16 + 0 + 4 + 2 + 1$$

$$= 407_{10}$$

$$627_8 = 6\cdot8^2 + 2\cdot8^1 + 7\cdot8^0$$

$$= 6\cdot64 + 2\cdot8 + 7\cdot1$$

$$= 384 + 16 + 7$$

$$= 407_{10}$$

$$407_{10} = 4\cdot10^2 + 0\cdot10^1 + 7\cdot10^0$$

$$= 4\cdot100 + 0\cdot10 + 7\cdot1$$

$$= 400 + 0 + 7$$

$$= 407_{10}$$

Let us compare the way we would count in three different systems:

Decimal Base 10	Octal Base 8	Binary Base 2	Tertiary Base 3
0	0	0	
1	1	1	
2	2	10	
3	3	11	
4	4	100	
5	5	101	
6	6	110	
7	7	111	
8	10	1000	
9	11	1001	
10	12	1010	
11	13	1011	
12	14	1100	
13	15	1101	
14	16	1110	
15	17	1111	
16	20	10000	
17	21	10001	
18	22	10010	
19	23	10011	
20	24	10100	

To see if you really understand counting, fill in the tertiary system using the symbols 0, 1, 2 where 0 is less than 1 and 1 is less than 2. You can check your result by looking at the answer on the last page.

III. Arithmetic Operations.

A. Addition

Addition raises a number to the next greater number a given number of times. Speed is achieved through memorization by the individual of the decimal addition table. In performing addition one thinks of the addend and augend and reads the sum from his memory.

Addition tables can be arranged for any symbol-positional number system; those for binary and octal systems are shown below. Here is an example of how two quantities (26_{10} and 58_{10}) can be added in the decimal, binary, and octal systems:

Decimal	Binary	Octal
26 - addend	11010	32
58 - augend	111010	72
84 - sum	1010100	124

Addition Tables

Binary

+	0	1
0	0	1
1	1	10

Octal

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	10
2	2	3	4	5	6	7	10	11
3	3	4	5	6	7	10	11	12
4	4	5	6	7	10	11	12	13
5	5	6	7	10	11	12	13	14
6	6	7	10	11	12	13	14	15
7	7	10	11	12	13	14	15	16

1. Binary

```
 11010
  1110
101000
```

Zero and zero gives zero. One and one gives zero with a one carry. Zero and one is one plus the one carry gives zero plus a one carry. One and one is zero plus the one carry gives a one plus a one carry. One plus the one carry gives a zero and the one carry.

```
 101010      101010
 110011      111111
1011101     1101001
```

Binary addition of two numbers is quite simple. However, the addition of more than two numbers complicates the manner of making the carry to the next column to the left. For a fairly long series of numbers, the carry (resulting from adding a column) may extend more than one column to the left. Witness the difficulty in trying to handle the carry when adding these binary numbers:

```
   11010
  111010
  101011
   10010
   11001
10101010
```

There are several ways to sum these numbers. We can add first to second; take this sum and add to third; take this sum and add to fourth; etc. This gives:

```
  11010    1010100    1111111   10010001
 111010     101011      10010      11001
1010100    1111111   10010001   10101010
```

This is a slow but sure (we hope) method.

2. <u>Octal</u>

```
362
157
541
```

Seven plus two gives one and a one carry. Six and five plus the one carry gives four and one carry. Three and one plus one carry gives five.

```
 157     157
 721     777
1100    1156
```

Try the following problems:

```
(1) 1011101    (2) 1000001    (3) 11111011    (4) 2463
    1010111        1010010        11110110        7534
```

(5)	5773	(6)	6435	(7)	1001010	(8)	11111111	(9)	4733
	4556		5246		1011011		10101010		3562
					10111101		11111000		2246
					111110		11010101		1144

(10) 46465
35725
64253
77667

B. Subtraction

Subtraction is the inverse operation of addition. The numbers used in subtraction have names as follows: The subtrahend is the number that will be subtracted from the minuend (number on top). The remainder is the result of the subtraction, thus:

7953 - Minuend
4132 - Subtrahend

3821 - Remainder

Subtraction reduces a number to the next lower number a stated number of times. In machine calculating, subtraction is more difficult than addition because of a physical process that must be reversed if the calculation is to be performed in a straightforward manner. Many of the devices used in machine calculation cannot be reversed easily, if at all. The complement method facilitates subtraction, providing the correct sign as well as sudden increase from zero. If a number C (the complement) can be found and added to a given number A, it will give the same result as subtracting the number B from A. Here, C is the complement of B. In the decimal system, two complementing methods exist: the 10's complement and the 9's complement. The 10's complement forms by subtracting the number B from 10^n where n is the number of digits available on the machine. For example, to subtract 426 from 782 in a four digit machine, the steps are as follows: The complement of 426 is 9574. (426 from 10000 = 9574). Adding 9574 to 782 yields 10356. The 1 in the fifth position does not appear (there being only four positions in the machine), so the result is 356, which is correct.

Machine difficulty arises in forming the 10's complement because all corresponding digits of the original number must subtract from 9, except for the farthest righthand non-zero digit, which must subtract from 10. This difficulty is overcome by resorting to the 9's complement.

The 9's complement is formed by subtracting the number from $10^n - 1$ (9999 if n equals 4) rather than from 10^n. Thus, each digit of the number is subtracted from 9, and the machine does not have to distinguish any special operations. Using the 9's complement:

The complement of 426 is 9999 - 426 = 9573
Adding 9573 to 782 gives 10355.
Here the 1 in the fifth position does not appear but is carried (end-around carry) to the first position and added to 0355, yielding 356, which is correct. Thus:

$$\begin{array}{r} 782 \\ \underline{9573} \\ 0355 \\ \underline{1} \\ 0356 \end{array}$$

The 9's complement requires one place to the left for a sign indicator. If the digit in this position is 0, the numbers following are positive; if 9, they are the complement of a negative number.

1. Binary

Binary subtraction follows the same rules, but with more convenience. When using complements, it is necessary only to change all 0's to 1's and all 1's to 0's. Machine methods accomplish this change easily. Binary subtraction, with and without complements, is shown in the following example - with decimal equivalents:

Without Complements

Decimal	Binary
$\begin{array}{r} 57 \\ \underline{23} \\ 34 \end{array}$	$\begin{array}{r} 111001 \\ \underline{10111} \\ 100010 \end{array}$

With Complements

Decimal	Binary
$\begin{array}{r} 057 \\ \underline{976} \\ 1033 \\ \underline{1} \\ 034 \end{array}$	$\begin{array}{r} 111001 \\ \underline{101000} \\ 1100001 \\ \underline{1} \\ 100010 \end{array}$

2. Octal

Octal subtraction occurs in a similar way. The inverse of addition or 7's complement can be used.

Without Complement

Decimal	Octal
57	71
23	27
34	42

With Complements

Decimal	Octal
057	071
976	750
1033	1041
1	1
034	042

Try the following problems:

(1)	(2)	(3)	(4)	(5)
110010	11000010	4635	2445	7777
101010	10111110	3477	1234	5432

C. Multiplication

Multiplication successively adds the same number a given number of times. But in practice, successive additions usually give way to memory (multiplication tables) to obtain the product of two numbers. Multiplication tables for the binary and octal systems are shown on the next page. The rules for multiplying can be observed by the way they work in the decimal systems; here 219 is to be multiplied by 45:

219	219		219
4	5		45
36	45	or	1095
4	5		876
8	10		9855
876	1095		

The multiplication process in any single radix consists of a routine of continually changing steps:

1. Select multiplicand digit
2. Select multiplier digit
3. Refer to the appropriate multiplication table
4. Obtain the value of the product
5. Add the preceding carry
6. Set down the right-hand digit in a position related to the position of the multiplier digit
7. Carry the left-hand digit
8. Repeat steps 1 to 7 until all multiplications have been performed
9. Add the digits set down

The sixth step involves the operation of shift, by which the significance of the position of a mark in a written number is maintained. In machine calculation, provision must be made for automatic shifting to obtain the proper number of positions for summing partial products.

Multiplication Tables

Binary

x	0	1
0	0	0
1	0	1

Octal

x	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	10	12	14	16
3	0	3	6	11	14	17	22	25
4	0	4	10	14	20	24	30	34
5	0	5	12	17	24	31	36	43
6	0	6	14	22	30	36	44	52
7	0	7	16	25	34	43	52	61

Try the following problems:

(1)	(2)	(3)	(4)	(5)	(6)
11011	1010101	10001	463	5776	423
101	10101	111	34	526	74

D. Division

Division reverses the process of multiplication; that is, it consists of repeated subtraction or determining the number of times (the quotient) the divisor can be subtracted from the dividend. An example is division in the decimal system:

```
          13
    25/  328
        -25 - - - - - 1
          78
         -25  - - - - - - - - 1
          53
         -25  - - - - - - - - 1
          28
         -25  - - - - - - - - 1
           3
                       _________
                         1    3
```

Note that subtraction continues without shift until a negative remainder results; at this point the appropriate shift occurs to the proper position. In the binary system, division proceeds in a similar manner:

```
                     1101
    11001 /  101001000
              11001
              100000
               11001
                 11100
                 11001
                    11
```

Of course, the octal system (or any other single radix system) follows the same steps for division of one number by another. The quotient of two numbers in most cases will not be an integral number. The radical point separates the integral number from the remainder; in the decimal system the radical point is called the decimal point. Referring to the formula for forming numbers, it will be seen that numbers to the right of the radical point develop as increasing negative powers progressing to the right: in the binary system they are powers of 1/2, 1/4, 1/8, etc.

```
       53
    7/ 463
       43
        33
        25
         6
```

Try these problems:

(1) $11101\overline{)10011101}$ (2) $111\overline{)100001}$ (3) $426\overline{)2465}$ (4) $777\overline{)11112}$

IV. Conversion of Whole Numbers from One System to Another

A. From Decimal to Octal

There are numerous ways of converting numbers from one system to another. For the sake of simplicity we shall only consider one method - the so-called remainder method. Other methods which can be found in the data processing publications are the "power reduction method" and the "iterative method", to name a couple.

To convert a number of one base into another system, the number must be divided by the new base <u>using the arithmetic operations of the original system</u>. The result is an integral quotient plus a remainder which will serve as the right-most digit of the converted number. The quotient is in turn divided by the new base, giving a new quotient and remainder. The second remainder forms the next digit of the converted number. This process is continued until a quotient of zero is obtained. The order of the remainders must be reversed to obtain the proper result. Let us look at an example: Convert 340_{10} to octal.

$8/340$	with remainder of 4	
$8/42$	with remainder of 2	$340_{10} = 524_8$
$8/5$	with remainder of 5	
0		

Convert these decimal numbers to octal:

(1) 100 (2) 1500 (3) 10,000 (4) 100,000 (5) 1,000,000

B. From Decimal to Binary

The same method is followed as for the decimal to octal conversion. Let's look at an example: Convert 349_{10} to binary.

$$\frac{349}{2} = 174 + 1$$

$$\frac{174}{2} = 87 + 0$$

$$\frac{87}{2} = 43 + 1$$

$$\frac{43}{2} = 21 + 1$$

$$\frac{21}{2} = 10 + 1$$

$$\frac{10}{2} = 5 + 0$$

$$\frac{5}{2} = 2 + 1$$

$$\frac{2}{2} = 1 + 0$$

$$\frac{1}{2} = 0 + 1$$

Therefore $349_{10} = 101011101_2$

Convert these decimal numbers to binary:

(1) 350 (2) 256 (3) 4096 (4) 1750 (5) 9999

C. <u>From Octal to Binary</u>

If we notice the relationship between our equivalents of decimal numbers in the binary and octal systems we will see a quick simple method of converting from octal to binary.

$$349_{10} = 535_8 = 101011101_2$$

If we break up the binary number into groups of three we get 101 011 101 which you can see is octal 535 if we evaluate each group. Therefore, to convert from octal to binary you simply write the binary equivalents from the octal digits. For example, octal 7531_8 = 111 101 011 001 = 111101011001_2. The intermediate step is not really necessary.

Convert these octal numbers to binary:

(1) 5745 (2) 1345 (3) 2376 (4) 123 (5) 77777

D. From Binary to Octal

You can see immediately to convert from binary to octal we would do the reverse of what we do to convert from octal to binary. Thus we group the binary number in threes and replace each three bits with their octal equivalent. For example: 100101_2 = 100 101 = 4 5 = 45_8. Again the intermediate steps are only put in for purposes of making clear the transition.

Convert these binary numbers to octal: (1) 1101011 (2) 11000001

(3) 1111110011010001010

E. From Binary to Decimal

We go back to our remainder method to convert binary to decimal. We divide the binary number by the binary equivalent of the decimal base (1010 for base 10) obtaining an integral quotient, plus a remainder. Write down the remainder separately. Then express this remainder as a decimal digit, the least significant digit of the answer. Divide the new integral quotient by the binary equivalent of the base, again obtaining an integral quotient plus a remainder. Express the binary remainder as a decimal digit and write each successive decimal digit to the left of the first remainder. Continue until an integral quotient of zero is obtained. For example, convert 1100101001 to decimal:

$\frac{1100101001}{1010}$	1010000 + 1001	= 9	
$\frac{1010000}{1010}$	1000 + 0000	= 0	Thus: $1100101001_2 = 809_{10}$
$\frac{1000}{1010}$	0 + 1000	= 8	

Convert these binary numbers into decimal:

(1) 1101011 (2) 1000000000 (3) 100100100100

F. From Octal to Decimal

We follow the same method for converting octal to decimal as we followed for converting binary to decimal. Remember that you are dividing by the octal equivalent of the base into which you are converting (which is 12 in the case of decimal 10). Convert 573_8 to decimal.

$$\frac{573}{12} = 45 + 11 = 9$$

$$\frac{45}{12} = 3 + 7 = 7 \qquad 573_8 = 379_{10}$$

$$\frac{3}{12} = 0 + 3 = 3$$

Another method using decimal arithmetic is given in Appendix I. Convert these octal numbers to decimal:

(1) 5774 (2) 123 (3) 1750 (4) 144 (5) 26

V. Conversion of Fractions from One System to Another

A. From Decimal to Octal

To convert from decimal fractions to octal fractions we make use of the following algebraic nicety.

$$N = a_{-1}x^{-1} + a_{-2}x^{-2} + a_{-3}x^{-3} + \ldots.$$

multiplying both sides of equation by x:

$$xN = a_{-1} + a_{-2}x^{-1} + a_{-3}x^{-2} + \ldots..$$

Thus if we multiply the fractional part by its base we will have the coefficient of the first item as the whole number. Likewise, you can see that if we took away this term and multiplied the remaining terms by the base we would get the coefficient of the second term, etc. Let's try an example: Convert 0.345_{10} to octal.

$$\begin{aligned} 8\,(.345) &= 2.760 \\ 8\,(.760) &= 6.080 \\ 8\,(.080) &= 0.640 \\ 8\,(.640) &= 5.120 \end{aligned} \qquad \text{Thus } .345_{10} = .2605_8$$

Convert the following from decimal to octal:

(1) .500 (2) .666 (3) .750 (4) .250 (5) .125

B. From Decimal to Binary

We would follow the same procedure as above. For example: Convert $.345_{10}$ to binary.

$$
\begin{aligned}
2\,(.345) &= 0.69 \\
2\,(.690) &= 1.38 \\
2\,(.380) &= 0.76 \\
2\,(.760) &= 1.52 \\
2\,(.520) &= 1.04 \\
2\,(.040) &= 0.08
\end{aligned}
\qquad \text{Thus } .345_{10} = .010110_2
$$

Convert the following from decimal to binary:

(1) .333 (2) .143 (3) .167 (4) .200 (5) .111

C. From Octal to Binary

We can see from the above that the octal-binary relationship applies not only to whole numbers but to fractions as well.

$$.345_{10} = .2605_8 = .010110_2$$

We can apply the same method for conversion from octal to binary for fractions that we use for whole numbers, namely, breakdown the octal digits into three binary digits that are equivalent.

Convert the following from octal to binary:

(1) .5247 (2) .4 (3) .6 (4) .2 (5) .1

D. From Binary to Octal

This is the same method that we use for whole numbers. Beginning at the binary point, arrange the binary number in groups of three and replace each group with its octal equivalent.

Convert from binary to octal:

(1) .0101010101 (2) .001001001 (3) .0010101 (4) .001100110011

E. From Octal to Decimal

We use the same method to convert from octal to decimal that we used to convert from decimal except that we multiply in the octal system using the octal equivalent for 10 (12_8). For example, convert $.423_8$ to decimal.

$$12\ (.423) = 5.276$$
$$12\ (.276) = 3.554$$
$$12\ (.554) = 7.070 \qquad \text{Thus } .423_8 = .5371_{10}$$
$$12\ (.070) = 1.060$$

Try converting the following fractions from octal to decimal:

(1) $.77_8$ (2) $.53_8$ (3) $.24_8$ (4) $.11_8$ (5) $.12_8$

F. From Binary to Decimal

We use the same method as above for the binary to decimal conversion, except that we would now multiply in the binary system and by the binary equivalent of ten (1010_2). For example, convert $.1000100011_2$ to decimal.

$$1010\ (.100010011) = 101.01011111$$
$$1010\ (.010111110) = 11.1011011$$
$$1010\ (.1011011\quad) = 111.000111 \qquad \text{Thus } .1000100011_2 =$$
$$1010\ (.000111\quad) = 1.00011 \qquad 5371_{10}$$

As you can see, this is the same problem binarily as the octal problem above, and it checks out.

Try the following binary to decimal conversions:

(1) .1000010 (2) .111111 (3) .00000001 (4) .101010 (5) .000110011

VI. Rounding Your Answers

The only question which should remain now is how many places you have to carry out an octal conversion to have it equivalent to the decimal number. The best way to consider this is to consider the logarithms of the numbers 2, 8, and 10. This would help us to see what is the relationship between them. Log 2 = .301, log 8 = .903, and log 10 = 1.000. To find the answer to our first question we would divide log 10 by log 8 which gives $\frac{\log 10}{\log 8} = \frac{1.000}{} = 3.3$ so we can approximate that there should be one more octal digit than decimal digit and four binary digits for each decimal digit.

VII. Answers

II. Counting

Tertiary number system (page 7)

0, 1, 2, 10, 11, 12, 20, 21, 22, 100, 101, 102, 110, etc.

III. Arithmetic Operations

Addition, Binary & Octal (pages 12 and 13)

(1) 10110100 (2) 10010011 (3) 111110001 (4) 12217

(5) 12551 (6) 13703 (7) 110100000 (8) 1101110110

(9) 14127 (10) 270554

Subtraction, Binary & Octal (pages 14, 15 and 16)

(1) $\begin{array}{r} 10000111 \\ \underline{1} \\ 1000 \end{array}$ (2) $\begin{array}{r} 1000000011 \\ \underline{1} \\ 100 \end{array}$ (3) $\begin{array}{r} 11135 \\ \underline{1} \\ 1136 \end{array}$ (4) $\begin{array}{r} 11210 \\ \underline{1} \\ 1211 \end{array}$

(5) $\begin{array}{r} 12344 \\ \underline{1} \\ 2345 \end{array}$

Multiplication, Binary & Octal (pages 16 and 17)

(1) 10000111 (2) 11011111001 (3) 1110111 (4) 20624

(5) 4002524 (6) 40164

Division, Binary & Octal (page 18)

(1) 101 (2) 100 (3) 4 (4) 11

IV. Conversion of Whole Numbers from One System to Another

Decimal to Octal (page 19)

(1) 144 (2) 2734 (3) 23420 (4) 303240 (5) 3641100

Decimal to Binary (page 19)

(1) 101011110 (2) 100000000 (3) 1000000000000

(4) 11011010110 (5) 10011100001111

Octal to Binary (page 20)

(1) 101111100101 (2) 001011100101 (3) 010011111110

(4) 001010011 (5) 111111111111111

Binary to Octal (page 21)

(1) 153 (2) 301 (3) 1763212

Binary to Decimal (page 21)

(1) 107 (2) 512 (3) 2340

Octal to Decimal (page 21)

(1) 3068 (2) 83 (3) 1000 (4) 100 (5) 22

V. Conversion of Fractions from One System to Another

Decimal to Octal (page 22)

(1) .400 (2) .5247 (3) .6 (4) .2 (5) .1

Decimal to Binary (page 23)

(1) .0101010101 (2) .0010010010 (3) .0010101010

(4) .0011001100 (5) .0001110001

Octal to Binary (page 23)

(1) .101010100111 (2) .100 (3) .110 (4) .010 (5) .001

Binary to Octal (page 23)

(1) 2524 (2) 111 (3) 124 (4) 1463

Octal to Decimal (page 23)

(1) .984 (2) .671 (3) .312 (4) .140 (5) .156

Binary to Decimal (page 24)

(1) .515 (2) .984 (3) .003 (4) .656 (5) .099

APPENDIX

OCTAL TO DECIMAL CONVERSION

Some people have difficulty performing octal arithmetic. There is a simpler method using decimal arithmetic.

To convert an octal number to a decimal equivalent we multiply each digit by eight starting at the left-most digit. We multiply the first digit by eight, add the second digit to this product, multiply this sum by eight, add the third digit to this product, multiply the sum by eight, etc.....until we reach the last digit. We add the last digit to the previous product but do not multiply this last sum by eight. Note the following example in which we wish to convert an octal 2436 to its decimal equivalent.

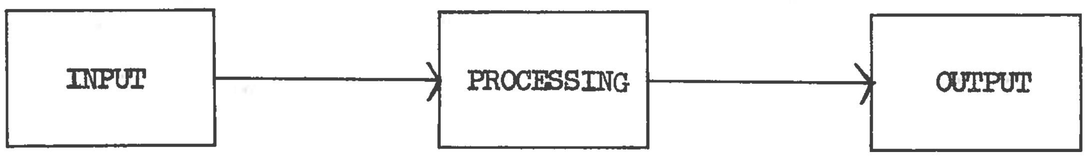
```
  2 4 3 6
 X 8
 ---
  16
 + 4
 ---
  20
 X 8
 ---
 160
 + 3
 ---
 163
X  8
----
1304
+  6
----
1310₁₀   ANSWER
```

You might try your ability to use this method using the problems given on page 22.

CHAPTER III

THE FIVE BASIC COMPONENTS OF AN ELECTRONIC DATA PROCESSOR

All data goes through three general phases in computer processing. The data must be put into the machine, processed internally by the machine and finally taken out.



These general functions or operations on data are accomplished through the interaction of five basic components found in every electronic data processing machine. These components are:

1. Input unit
2. Output unit
3. Storage unit
4. Arithmetic-logic unit
5. Control unit

These components (called the "hardware" of a computing system), efficiently combined, can handle the following general computer operations: data conversion to electrical quantities, addition, subtraction, multiplication, division, comparison, data transfer, data storage, operating sequencing and timing, yes-no decision making and data conversion to needed or desired form. Let us now discuss these basic components indivdually and their inter-relationship.

Data to be used or produced by a computer is usually recorded on some physical substance or MEDIUM. Not only does this allow the most economical handling by the computer but it makes it possible to compile, check, verify and correct the data prior to machine processing. Subsequent to processing, data produced by the computer on some medium can be checked, verified, corrected, analyzed and finally stored for future use.

Early data processing systems depended on the punched card and paper tape mediums. Generally, the modern day system uses magnetic tape as the primary medium with the punched card, punched paper tape and printed paper as secondary mediums.

A computer would be worthless to man unless "data" could be "fed" into it and "extracted" from it after being processed. These functions are called data INPUT and data OUTPUT. "Peripheral" devices (input and output devices) have been developed which will feed data to the computer, others to extract data and still others which can function as both input and output units.

Let us begin our discussion of the five basic components of a data processing system by introducing a few of the available input devices.

1. INPUT DEVICES

<u>CARD READER.</u> This device is capable of "reading" or interpreting data as represented on punched cards (where alphabetic characters, numeral and special characters are represented by a set configuration of holes punched into a certain area of a card). It then translates these characters into electrical pulses which are transmitted to the unit of the computer where they can be electronically sorted until processed. The speed at which this unit will operate varies from 250 to 1000 cards per minute. Even at the highest speed this device is classified as a medium speed input device.

Most card readers operate electromechanically although some have been developed to read cards photoelectrically. The electromechanical card reader has feelers (usually wire brushes) which are used to detect holes in the card. The most common devices read either one column of a card at a time or one row at a time. However, there are devices where the entire card is read at one time using 960 brushes.

An important feature of the card reader is that design characteristics give it the ability of verifying characters before they are transmitted to the computer. Many different checks can be made during the reading process (like checking to see if an invalid number of punches appears in any one column) to determine the existence of typographical errors in the card or errors caused by machine malfunctions.

<u>PAPER TAPE READER.</u> Data can be represented by punches in paper tape in a way that is very similar to the method used with punched cards. The paper tape reader interprets these holes,

verifies them as being valid and then transmits them as electrical pulses to the machine area where they are held or stored until processed. These devices operate from speeds of 100 to 2,000 characters per minute thus placing them in the medium speed class. Most modern paper tape readers operate on photoelectric reading pinciples, although the early readers were almost exclusively mechanical pin-readers.

<u>TYPEWRITER.</u> A special built typewriter may be used to enter data in the computer. Because its input speed is dependent upon the speed and accuracy of a human operator, its primary use is to enter program parameters (variables) or special instructions to the computer. It is considered to be a slow speed input device.

<u>CONSOLE.</u> The computer console switches, keys, dials, etc. (depending on the computer) can be used to input small amounts of information into the computer. Again, because the input speed is dependent upon the speed and accuracy of a human operator, it is used basically for changing instructions in a program or ordering the computer through special operations. It is classified as a slow speed input device.

<u>MICR.</u> The function of the magnetic ink character recognition unit is to read and interpret special characters which have been imprinted with magnetic ink on documents (most commonly bank checks) and to transmit these characters as electrical pulses to the computer. This particular device has revolutionized the banking industry in recent years. Checks and other bank documents can be processed in a fraction of the time that was required to do the same kind of processing manually or with other kinds of equipment. It is classified as a medium speed input device.

<u>OPTICAL SCANNER.</u> This device is capable of reading and interpreting specified fonts (shape and size of letters) and transmitting them as electrical pulses to the computer. It is still too new a device to categorize, but its importance can be emphasized by its recent adoption by many in the Retailing Industry. Several retail department stores use them to interpret data (relative to daily sales, returns, losses, commissions, inventory controls, etc.) taken from the numerous cash register prepared paper tapes in the store.

<u>MAGNETIC TAPE UNIT.</u> Computers, when first developed, achieved speeds of five or six operations per second. Because so much emphasis was placed on internal processing speed, gradual improvements in the technology raised the speed to over a million operations per second. However, such tremendous operation speeds left the processing units of the computer system "waiting" for data because input devices such as the card reader and paper tape reader could not feed data fast enough. Thus in the middle 1950's there was a flurry of interest in the magnetic tape unit - a device capable of transmitting large volumes of data at high speed to the computer.

Magnetic tape used in computer systems is similar in principle to the tape used in home tape recorders except it must be of a particularly high quality. The tape is a tough and wear resistant plastic (about as thick as cigarette paper) which is coated with a ferromagnetic material. The representation or recording of characters occurs in this ferromagnetic coating. Magnetic spots (called bits) placed or magnetized on particular areas of the tape are used to represent data characters similar to the way holes in a punched card or paper tape represent characters.

The tape unit is capable of "reading" or interpreting these magnetic bit configurations and transmitting the correct electrical pulses to the computer at a high rate of speed. The rate of character transmittal will depend upon the capability of the unit and the density (number of characters per inch) of characters on the tape. The higher the density and the faster the operating speed of the unit, the higher is the rate of information read from the tape.

The "reading" of information is accomplished by the tape being transported from one reel of tape to another past a "head assembly" which can recognize the presence or absence of magnetic spots on the tape. Generally, this rate of movement is from about 36 inches to 112 inches per second. Depending on the character density, this means a general flow of from 7,200 to 90,000 characters per second. Starting and stopping the tape is so rapid that a small amount of slack tape must be maintained so that the tape will not be damaged or torn. In general, there are three methods of providing slack tape: Idler, vaccum loop and weighted loop. The most common method is the vacuum loop where a loop of tape from each of the two reels on the tape unit is held in vacuum columns.

As with other input devices the magnetic tape unit is built to recognize and check for errors (like analyzing the signal strength of data recorded on tape) and to go through certain automatic correctional routines (such as cleaning dust from sections of tape and attempting to re-read these sections). Generally, these error operations and other related operations are controlled by the computer.

The magnetic tape unit is classified as a high speed input device surpassing other devices like the card reader. The magnetic tape unit, like any other mechanical device, cannot input data to a computer at a speed that approaches the operating speed of the computer.

2. <u>OUTPUT DEVICES.</u> There are a variety of output devices to produce data in almost any needed form for either human consumption or machine consumption. Some of these devices do not use a medium (like punched card) but display data on screens or tubes. We will discuss only of these special purpose output devices since we wish to emphasize the more common output devices in this section.

CARD PUNCH. Basically this output device falls into two groups of machines. A card punch operated by a human operator (known as a key punch) can be used to take raw data (written reports and information) and output this data on cards in a machinable form where characters are represented by punches in the card. The other group of card punches are those associated with an electronic data processing system. In this case, the card punch is under the control of the computer or the program being run on the computer. Information being held by the computer in the form of electrical pulses is transmitted to this unit where the pulses are translated to punches in a card.

PAPER TAPE PUNCH. There are also two groups of tape punching machines. One group, like the flexowriter, is a manually operated key punch board device which produces as output a paper tape where particular configurations represent characters. The term paper tape punch most commonly refers to those devices in a data processing system under the control of the computer. In such a system, data as represented by electrical pulses in the computer is transmitted to this device where the pulses are translated to punches in a paper tape. This device is classified as a medium speed output device.

TYPEWRITER. A special typewriter like device was used extensively to output data in the early data processing systems when speed was not so important a factor. Its earlier function has been replaced by the new generation high speed printers. In current systems it is commonly used as a communication device between the computer and the computer operator or the programmer.

CONSOLE. Through the console switches, dials and lights small amounts of data can be displayed enabling the computer operator or programmer to analyze the overall operation of a program.

PRINTER. Since most output data must be a form readily acceptable for human consumption, the high speed printer is usually one of the basic output devices found in a data processing system. Whereas the typewriter prints one character at a time, the printer can print one or more lines at a time. Printers differ in their rate of speed and methods of printing. Many printers are similar to the IBM Model 1403 printer in the IBM 1401 system where the type is contained on a continually moving belt (a series of complete alphabetic, numeric and special characters on one belt). Data characters in the form of electrical pulses are transmitted to the printer where a hammer is activated to strike a particular character and print it in the correct position of a line. Other printing methods include a series of rotating wheels (where each wheel contains the desired characters) or a matrix box (where different prints of the matrix make up all the desired characters). Printing speed varies from

about 150 to 9,000 lines per minute, but only those printers that operate above 600 lines per minute are considered high speed. Since even the highest rate of the printer is well below the output rate of the computer, large volume printing operations will usually be confined to the smaller and less expensive data processing systems (like the IBM 1401 system).

MAGNETIC TAPE UNIT. Data in the computer or the storage unit can be transmitted to the magnetic tape unit where character representations are written out on the magnetic tape in the form of magnetic spots. The output operation is accomplished when the tape is moved from the left reel through the left vacuum column across the head assembly (contains both a read and write head) through the right vacuum column to the right reel. During the write operation and before the tape reaches the head assembly an erase head erases all previously written information. This allows a tape to be used over again until normal wear reduces the quality to a level unacceptable in a data processing system.

As in the input operations, the tape unit is capable of verifying and checking as it writes or outputs data. Classified as a high speed output device, the general output speeds of the magnetic tape unit are the same as mentioned in the description of input devices.

We have briefly described some of the more common input and output devices in an electronic data processing system. Some of these units are single purpose as they are used to either input data or output data. In some systems an input and output device may be combined in one cabinet (like the IBM Model 1402 card reader-punch in the IBM 1401 system) for convenience and "hardware" sharing. A few of these units are very versatile in that one unit (i.e., the magnetic tape unit) can function as an input unit (when reading) and as an output unit (when writing).

In a computing system almost any combination of input and output (I/O) units can be used in accordance with the physical limitations of the hardware and the demands being made on the system. For example, a possible configuration for an IBM 1401 system would include six magnetic tape units for input and output, a card punch and printer for output and a card reader for input operations. Some computing systems are oriented toward one medium, and therefore, the configuration of I/O devices would reflect this orientation. In general the less expensive computing system will tend to be a punched card or paper tape system while the more sophisticated would depend on magnetic tape.

In the more sophisticated, faster and expensive systems like the IBM 7090 (its cost is approximately $2,800,000.00 compared to the IBM 1401 cost of about $125,000.00) the magnetic tape unit is the primary input and output device. All data to be fed into the machine must be prepared for use by converting it to magnetic tape away from the 7090 system (this is called OFF-LINE processing) as opposed to all data being handled "as is by the 7090 system (ON-LINE processing). Because of the requirement for conversion from one medium to another, special devices have been built to convert paper tape to cards, paper tape and cards to magnetic tape, etc. This type of data conversion processing is usually done off-line. All data to be used by a computing system will usually be on a medium most easily and economically handled by that particular computing system.

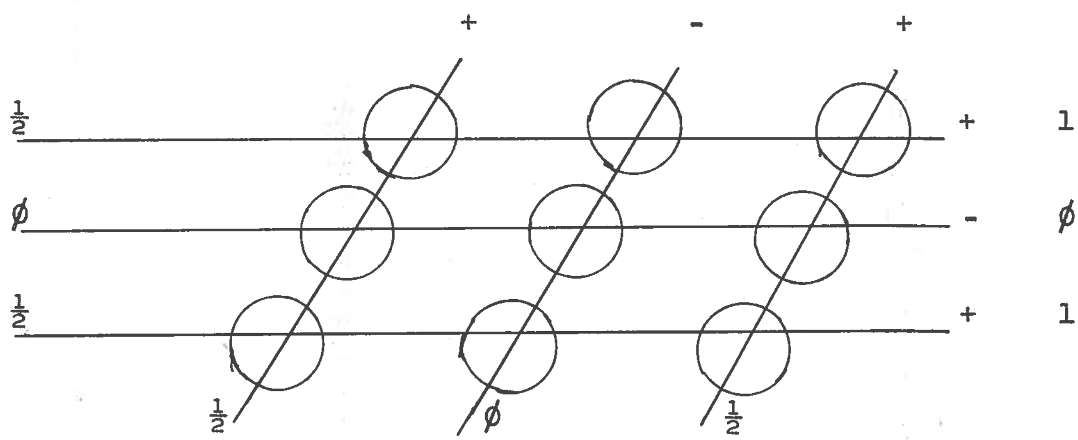
3. <u>STORAGE UNIT.</u> The term "mechanical brain" or "electronic brain" was first applied to the electronic computers at their conception and subsequent development stages. These machines do in fact simulate the processes of the human brain and have the capabilities of performing data processing operations as man does (like the ability to perform arithmetic operations and to make logical decisions). However, computers are far removed from the theorectically possible "thinking and reasoning" machine having all the characteristics and abilities of the man because of their inability to sophisticately coordinate data without the aid of man. The last three basic components of electronic data processing system (storage, control and arithmetic-logic units), though recognizable as mechanical imitators of different sections of the human brain, are still limited when compared to the overall tremendous capabilities of the human brain.

Just as the human brain has the ability to store facts or data, so the electronic computer can retain data in a unit called STORAGE and MEMORY. In the electronic data processing system, storage can be classified as either INTERNAL STORAGE or EXTERNAL STORAGE. For example, when a man retains facts in his head or brain, he is using his "internal storage" capacity. When the same man decides that there are too many facts or the facts are too involved to remember, he might put some of these facts down on paper. Thus the paper can be thought of as "external storage". External storage is separate and is used to add to the capacity of internal storage. Now, let us discuss these two types of storage.

In an electronic data processing system the internal storage unit is one of the basic units of the system. It is a physical part of the system and is controlled by the system. Historically, the development of internal storage has gone from electro mechanical devices such as relays and counters, sonic delay lines, and vacuum tubes to magnetic core storage which today is the most common type of interal storage.

MAGNETIC CORE STORAGE. Just as transistors have made it possible to reduce the size of all the components of the electronic data processing system, the development of magnetic core storage has greatly reduced the size of the storage unit and has at the same time increased its efficiency and reliability. A magnetic core is a tiny doughnut-shaped ferromagnetic ring, a few hundredths of an inch in diameter. Aside from its compact size, the important characteristic of the core is that it can be magnetized in a few millionths of a second and, unless deliberately changed, it retains its magnetism indefinitely. Thus a power failure in a system using this storage method would not effect memory (a common characteristic of the earlier systems using other methods).

The following is a simple diagram of magnetic core storage:



As seen in the illustration, small rings of ferromagnetic material are strung on wires. When half the current needed to magnetize a core is sent through two wires, only the core at the intersection of the wires is affected. The polarity of the magnetized core or ring may be either in a clockwise direction or a counter clockwise direction. Thus these two possible states or conditions of a core may be used to represent the binary digits zero (0) and one (1). By combining a number of these cores in a logical group, alphabetic characters and digits of almost any numbering system may be coded in the binary mode. The grouping of cores is accomplished by the stacking of planes of cores (called an array). The magnetization of all cores which occupy an identical position in their plane throughout the array would give us a coding pattern to represent alphabetic characters and numeric digits. In some cases, the coding pattern would be very similar to that used in seven channel magnetic tape.

Magnetic core storage becomes quite an important storage device because it comes closer to meeting the theorectically desirable storage criteria of capacity, reliability and access than any other storage device.

Earlier devices in particular could not meet these criteria. For instance, the electromechanical devices were subject to mechanical failure, were large because of all the needed wiring and very slow. Tube storage was a huge device with the many unreliable and short-lived tubes, many feet of wiring and a tremendous air conditioning requirement. When using tube storage, a power failure would result in the loss of all data in storage. In contrast, magnetic core storage made in matrices or planes is compact and any number of planes can be joined in logical arrays to meet almost any storage capacity requirement. If there is a power failure, the magnetizism or data representation is retained. With magnetic core storage, one has RANDOM ACCESS or rapid access to any part of storage.

Matrices similar to magentic core storage are being built with other materials which are simplier and cheaper to manufacture since cost has been one factor in the use of magnetic core storage until now. New methods or devices like THIN FILE electrostatic storage are becoming important and may soon replace magnetic core storage (or similar matrice storage). In spite of all our technological sophistication, memory remains a major computer problem.

A "word" in the electronic calculating machine is the basic unit in storage having meaning by itself. The term "word" can mean either a constant number of characters or a constantly varying number of characters. Let us now examine these different meanings.

We have mentioned in this text that there are basically two different applications for our electronic calculating devices. They can be used as computing machines or data processing machines. Those machines leaning toward computing applications (generally scientific and mathematical problems) are usually Fixed word length machines. This means that a "word" in this machine is set at a certain number of characters or bits. For instance, the IBM 7090, a computing machine, has a 36 bit "word". Storage in this machine is broken up into words and each word has a specific ADDRESS referring to an exact location in storage. If a fixed word length machine had 500 locations, these locations might be addressed by the numbers 001 through 500.

The group of electronic calculating machines that are most applicable to processing data or handling business problems are usually variable word length machines. By this we mean that the number of characters treated as a "word" in this machine (data processor) will vary from one to the total number of characters that can be stored in memory at one time. In this type of machine, like the IBM 1401, only one character can be stored in each location. Therefore, an "address" in this machine will refer to the location of one character.

Why were electronic calculating machines developed to operate as computers or data processors? This two-field specialization

occurred because in the early development stages of electronic calculating devices the budding technology of building these machines lacked the sophistication necessary to combine both applications in the operation of one machine.

The variable word length machine usually operates on data serially (one character at a time) thereby reducing its cost because of the simplicity of operation but at the expense of speed. Word separators are used to separate words; therefore, any length word can be handled. Contrasting with the DATA PROCESSOR is the COMPUTER which operates on data in a parallel fashion - one word at a time. Speed is the result of this type of operation. Judging from the make-up of these machines it is easy to see why data processors are best suited for business operations. By building them as cheaply as possible (small memories, serial operation and fewer basic machine operations, etc.) and teaming them with high speed input and output devices, the data processor type of electronic computer is available to "many" for almost any kind of daily processing from inventory accounting to payroll procedures. The computer, on the other side of the same fence, is costly because it needs more core or memory, more speed, (using parallel operations) and more technologically complicated components.

Evolution would have it that these two "types" of machines would some day be integrated into one. Today our technology has advanced so far as to produce symbolic languages which enable us to use a "business oriented language" like COBOL on a scientific oriented computer like the IBM 7090 and to produce a "scientific oriented language" like FORTRAN which can be used to program the operation of a data processor like the IBM 1410. Thus, the classifications "COMPUTER" and "DATA PROCESSOR" tend to have less meaning. Also, developments like the IBM 360, which is a machine applicable to both business and scientific operations, will bring us a general machine to meet any of the requirements we might place on it.

<u>EXTERNAL STORAGE.</u> Those storage facilities external to the central processor which hold information before and after it has been processed within the computer are called external storage. Information in external storage is in a form suitable for direct entry into the computer. The most common devices used for external storage are the magnetic disk, magnetic drum, and magnetic tape units.

<u>MAGNETIC DISK.</u> This storage unit resembles the latest in juke boxes. Data is recorded on metal disks, about two feet in diameter, that are coated on both sides with a ferrous oxide recording material (a material easily magnetized). The disks are mounted on a shaft so that the disks (which are separated from one another) can be rotated at a speed of about 1200 RPM. Information is recorded on these disks in the form of magnetic spots (similar to magnetic tape) in concentric tracks on each side of the metal disk.

A forked access arm (one or more) mounted at the side of the stack of disks can move to any desired track on any disk. Each arm has a read-write assembly to read or write on either surface of a disk. The more access arms a unit has the closer that unit gets to having random access to the data on the disks.

Data can be read repetitively from the disk without loss but each time new information is stored in a track it erases the data formerly stored there.

The latest adaption of the magnetic disk is the magnetic disk file. The actual disk may be removed from the unit and replaced at any time. This means that data (maybe in form of master files) for several large problems can be stored permanently on the disks and placed on the unit only during the operation of the specific program to handle the related problem.

MAGNETIC DRUM. This external storage device consists of a revolving cylinder, the surface of which is coated with a material that can be readily magnetized to hold data very similar to the method used on magnetic tape. The cylinder is divided up into sections (channels) and each of these channels is made up of some number of tracks necessary to represent a character (alphabetic or numeric) and to provide efficient operation. For speed of operation, each channel has a read-write assembly head. The drum is rotated at a constant speed past the read-write assembly head thereby giving excellent access to any given point of the drum. Because the distance between the read-write head and the drum is constant, higher operating speed is more attainable on the drum than magnetic tape. However, the bit density is less on the drum than the magnetic tape unit. The drum has proven to be a convenient and versatile medium for storage with its storage capacity limited only by its diameter and length. All input and output efforts to this storage unit are coordinated, and positioning is maintained exactly through the use of a special track called the timing track in each channel.

Just one magnetic drum or disc can provide increased storage capacity for the computer system in excess of 30 million characters. The next important feature is that using either of these units in a system you have the capability of random addressibility. This means that records (a data group) can be of related transactions prior to machine processing. A useful application might be the keeping of master records of a company on magnetic disc or drum so that these records can be updated continually throughout the day. This would mean that the company officers would have at hand an up-to-date report on the affairs of the company.

Neither of these units have true random access, as does Magnetic Core storage, because they are Electro-Mechanical. Mechanical speed can never match electronic speed.

4. ARITHMETIC-LOGIC UNIT. The arithmetic-logic unit is that unit where processing in an electronic data calculator actually takes place. It is usually made up of special registers (temporary storage devices) and other circuitry which enables it to accept data from storage (under the control of the control unit) and perform arithmetic and logical operations on data.

Usually all arithmetic operations such as addition, subtraction, multiplication and division have been reduced to basic addition to simplify the circuitry necessary to accomplish the given process and to take advantage of the efficiency of the binary mode that most computers use today. All of these operations are performed at the direction of the control unit which sets up the correct action and makes the data available. Following the completion of the assigned task, the results are made available to the control unit.

Logical functions can also be carried out in the arithmetic-logic unit. Circuitry enables this unit to determine the relative magnitude of a number or the relationship between two values. Information of this sort, furnished to the control unit, enables the programmer to program decision making capabilities into his program. The arithmetic-logic unit provides the programmer with most of his analytical tools.

5. CONTROL UNIT. This unit, which corresponds to a large switching station, is what makes an automatic calculator automatic. It has assumed all the required functions of a human operator. It takes its orders from the computer program and implements these orders in a logical fashion. The control unit accepts instructions from memory or storage, interprets instructions and sets in motion those forces that will actually perform, in the proper sequence, the designated operations. At the same time, it keeps account of where in storage the next instruction to be interpreted can be found, subsequent to completion of its present operations.

These functions are accomplished through the instruction decoder and control generator. The instruction decoder circuits are designed to recognize that a coded combination of pulses represents an actual instruction. Instruction recognition brings about a unique set of actions by the control generator.

Each instruction is made up of an operation part and an address part. The operation part of an instruction is interpreted by the instruction decoder during the instruction cycle of operation. The address part, which may actually contain one or more addresses, is used by the control generator during the execution cycle of operation

There is a great deal of overlapping of the control units circuits because a number of instructions require similar operations on the basis of their interpretation. This helps to simplify the control unit and to keep its size compact.

NOTES

CHAPTER IV

HARDWARE AND SOFTWARE

The term "hardware" refers to the physical components, input-output devices, and other pieces of equipment that make up a system, as might be expected from the common usage of the word. On the other hadn, "software" is a new word in the English language. It is used to identify the many detailed operations involved in the feeding of instructions to the computer equipment.

Concurrent with the development of computers there has been a continuing effort to reduce the labor involved in preparing a program of instruction to be followed by the computer. This effort has led to the development of "automatic programs". The term "automatic program" is difficult to adequately define, but in general is used to identify a program of instruction which translates the notational representation of a problem from a first form into a second form. Usually, the first form would be some format or notation developed for use by a programmer. It would emphasize ease of handling the various logical steps to be followed in the program. The second form of notation would be directly machine oriented and specifically determined by the computer characteristics. It would also involve the assignment of addresses and other features necessary for the solution of the problem.

The concept of automatic programming has been extended to the point where the individual using the computer need know relatively little about the computer. Automatic programming may be limited to specific rules and codes for a certain machine or may be wide enough to allow problem solving using a generalized set of rules applicable to any computer for which there is a suitable automatic program for performing a translation, from the expression of the problem to a program of instruction that the computer is able to execute.

The complexity of preparation of an automatic program increases in relationship to the generality of use. The creation of an automatic programming system for use with a complex computer is in itself a very large task requiring many man-years of work.

The general usefulness of automatic programming has become a very important facet in the manufacturing and sales of computer "hardware". Computer manufacturers have adopted the practice of preparing suitable programming systems or automatic programs as part of each new computer system. When in the market for a "hardware" system, the potential customer now evaluates the machine, in part, by estimating time factors in the solution of certain problems using the manufacturer's "software package".

In summation, we can say that "software" generally refers to the automatic programs used with the computer complex or "hardware". The term "software" is also used to signify any program or set of programs, which may or may not be "automatic" in nature; that is, available for general use from a manufacturer or from other sources.

MNEMONIC NOTATION

By itself mnemonic notation (i.e., any notation which tends to assist the memory.) is not considered to be a scheme for automatic programming. Nevertheless, mnemonic notation is an important feature of many automatic programming techniques and can be used to great advantage with even the most elementary of programming aids.

In the manual preparation of a machine language program without mnemonic notation, the programmer would likely write out the words (multiply, jump, etc.) that specify the functions to be performed by the various instructions in the program. Of course, the use of abbreviations is well known and the programmer may abbreviate in any way he sees fit. The abbreviations may be a resemblance to the original words, such as MPY or MUL for MULTIPLY, for example, and the abbreviations could then be said to have mnemonic characteristics. Although mnemonic notation usually does involve abbreviations (or using a single word in place of several words usually required in a conventional English language statement), the abbreviating is not the important feature of the mnemonic notation concept. In fact, in some instances, an objective is to eliminate any features of the notation that would require the programmer to learn anything whatsoever about the notation itself.

The important feature associated with mnemonic notation, as the term is usually used, is that the notation is accompanied by a computer program that will translate the notation into whatever notation is used by the computer. In an elementary automatic programming system, a notation such as MPY could be punched in a card with the code used for alphabetic notation in some specified card punching machine. The 1's and Ø's as represented by the array of holes punched for the letters M, P, and Y would have no direct relationship to the 1's and Ø's used in the operation part of the computer instruction for multiplication. The 1's and Ø's as punched in the card may nevertheless be entered into the computer, and the translation program can then be used to prepare the instruction as required for execution by the computer. The details of the translation program would depend on the characteristics of the computer. Conceptually, it would be as easy to translate multiply as MPY, but a greater amount of time would be required for punching the card, and if the word length within the computer is exceeded, the translation program would be more complicated and less efficient.

The use of abbreviations having fewer than three letters would probably not be classified as "mnemonic" except in systems capable of executing only a few different instructions although, of course, the automatic translation from whatever notation is used to the notation by the system itself is still possible.

In the more advanced automatic programming techniques, mnemonic notation has been used in more elaborate ways. For example ADD "a" and "b" to PRODUCE "c" may be exactly what the programmer writes at some point in the program (a and b having been defined previously). In a three-address or a four-address system, the example just given may correspond to a single instruction. In a one-address system at least two instructions would be required to perform the specified function, and in the general case a function specified in this way may require many instructions. Nevertheless, if the meaning of the statement is unambiguous, the automatic translation from the statement to a machine-language program is possible. Some automatic programming techniques are based on this possibility with a major objective being the use of an established written and spoken language to prepare programs in a manner that will minimize the knowledge of programming a person must have to prepare programs.

The objective set forth in the previous paragraph would be valid if the translation program needed to achieve the result were not so lengthy and time-consuming. When data processing is considered on an overall basis, the important ultimate objectives are usually to minimize the total costs, the processing time, or some combination of cost and time. Because of the expense of preparing the translation program and the expense (caused by the time consumed by the computer) at performing the translations, and because of the possibly inefficient ultimate machine language programs that result, the minimization of programmer effort is not necessarily of fundamental importance. The alternative is tc develop a notation that strikes some sort of a balance between having mnemonic characteristics and having characteristics that allow efficient translation to a machine language program that is itself efficient.

Opinions differ widely -- in fact, the matter could be said to be highly controversial - on the subject of notations that are the most desirable for programmers to use. Literally hundreds of different sets of notations have been developed, each one requiring a different translation program for each computer with which it is used.

THE ASSEMBLY PROGRAM

In the preparation of programs on a purely manual basis, the assignment of addresses at which the various instruction and data words are to be stored is quickly found to be a major effort and source of errors in spite of the apparently elementary nature of the task. For all but the very simplest of programs, a programmer is seldom able to prepare a program that accomplishes the intended result by doing nothing more than writing a list of instructions. At least a few revisions in the first draft nearly always seems to be necessary. Also, when using previously prepared subprograms, as is commonly the case, the subprograms cannot as a rule be complete in every detail because the locations in which the subprograms are to be stored must be assigned. If there are any jump instructions within a subprogram, the addresses of the jump instructions must be adjusted in accordance with the location at which the subprogram is stored (unless relative addressing is available). The instructions for entering and leaving the subprograms may also be dependent on the locations of the subprograms.

For long and complex programs that may involve many subprograms, the correct assignment of all addresses and the corresponding determination of the address parts of all instructions is tricky enough even when no revisions in the program are necessary. When the program MUST be modified, even by as little as inserting one instruction, the reassignment of addresses is a task that invites programming errors.

For example, when inserting an instruction, the addresses of all instructions and data words at higher numbered addresses would ordinarily be increased by one, and all instructions that refer to the relocated items must have their address parts correspondingly altered whether they are stored at lower or higher numbered address positions in comparison with the inserted instruction. Instructions having relative addresses need not be changed unless the relative address is such as to span the inserted instruction. With programs having thousands of instructions, the insertion of only one instruction may clearly require an extensive amount of work.

In some cases it may be impractical to modify the program in the elementary way just suggested. Instead, a part of the program containing the inserted instruction may be removed to some remote, but available, portion of storage with jumps being made to and from this portion as required. The total amount of work required for such a modification might be very much less, but the character of the alterations would obviously be more complex. The term "patching" is often applied to program alterations made in this general manner. When a large amount of patching has been done, the resulting program may be error-free, but it might be highly inefficient. After its peculiarities are forgottn, any subsequent modifications that may be found to be necessary then present even more serious problems.

Instead of assigning addresses and determining the address parts of instructions in a direct manner as implied in the preceding discussion, an alternative approach is to assign a symbol to each instruction or item of data to be stored. For the most part the symbols may be selected randomly except whenever a group of instructions or a group of items of data must be stored, at addresses that bear some relationship to each other. For example, in an algebraic problem, a variable which may be known as x may be assigned the symbol x in preparing the program. The specific address at which x is eventually stored is really irrelevant to the programmer. One of the exceptions would be in a sequence of instructions such as in a subprogram where the instructions must be stored in successively higher numbered addresses because of the manner in which the address counter functions in the computer. The sequence of instructions might then be assigned symbols such as a, a+1, a+2, and so on. The address parts of the various instructions as prepared by the programmer would not be actual addresses but would be the symbols as assigned. One-address examples of such instructions would be ADD x and JUMP a+2.

Now with the symbols assigned as described in the previous paragraph, a program can be prepared in independent portions. The portions can then be assembled with the symbols being replaced by actual addresses in a routine manner. Of course, anything that can be reduced to a fixed routine can itself be performed by a suitable computer program. A program for assembling portions of a program, assigning addresses, and determining the address parts of instructions in the manner suggested is called an "assembly program." If a program is to be modified after it has been assembled, the modifications may be made relatively easily on the instructions as written by the programmer, although a complete reassembly by the assembly program may be necessary.

Assembly programs represent one of the earliest of the important concepts that could be placed in the category of automatic programming. Assembly programs have been widely used and are still important although they can be replaced to a large degree by more sophisticated and more comprehensive techniques. Some automatic programming techniques employ the concept of the assembly program as the last step in the translation of a program from some POL to machine language, although the person using the POL need not be aware of the fact. Also, the preparation of the translating programs themselves is often done with the aid of assembly programs.

Probably the major feature of assembly programs that could be called a disadvantage is that there is not so much of a suggestion of relieving the programmer of a need to know the details of the machine instructions. The development of the assembly program itself is an extensive task, especially when done for a complex system. In favor of assembly programs is the fact that, because the language is essentially machine language the attainment of efficient program problems is more practicable than with some of the more sophisticated automatic programming methods.

From the standpoint of systems design, assembly programs do not appear to have had any great influence although undoubtedly the designers of specific systems have modified details here and there in response to suggestions of programmers who had been working with assembly programs.

POL

The use of different sets of notations or mnemonic codes for the writing of computer instructions has resulted in a situation where there are literally hundreds of different sets. These each require a special translation program and are generally restricted to a single machine. Much thought has been given to the possibility of the development of a universal notation that would be suitable to all people for all problems, but as might be expected, little progress has been made. Instead, a long list of "problem-oriented languages" (POL's) are appearing. Each POL is adapted to a certain restricted range of problems such as purely mathematical problems, business accounting problems, machine tool control problems, bridge structure problems, land surveying problems, and numerous others. Each POL has characteristics that are mnemonic for the people engaged in the field for which the language is intended.

At the present time the most widely accepted POL is one known as ALGOL and is intended primarily for purely mathematical problems. (See Appendix A, An Introduction to ALGOL)

COBOL is a POL that was developed with emphasis on being mnemonic to a layman and is intended for business applications. Comparatively, COBOL is quite close to conventional English and is gradually becoming more widely adopted in spite of its having received a substantial amount of criticism and even ridicule. If nothing else, COBOL and like languages have sales appeal in business circles.. (See Appendix B, COBOL, and Appendix C, A Simplified Approach to FORTRAN)

Most POL's are developed under the realization that from a practical standpoint the programmer must work with a finite and actually rather small number of different symbols in spite of the unlimited number of combinations (each representing a different symbol) of 1's and Ø's that can be transmitted to a computer as input. Often the number of different symbols is no greater than the number of symbols found on a typewriter keyboard. In fact, a conventional typewriter keyboard is often accepted as a fundamental limiting factor inasmuch as the programmers, who may be numerous and widely distributed geographically, will likely want to use such a readily available and relatively inexpensive instrument to record their programs. Even for POL's where more symbols are employed than can be placed on a conventional typewriter keyboard, the number of symbols actually used is seldom greater and is still usually limited to the number available with a keyboard instrument of some sort.

The symbols actually used in most POL's also correspond closely to the symbols found on a conventional typewriter. In

particular, most POL's make use of the 26 letters of the alphabet, both upper and lower case, and also the ten decimal digits. Further, the most commonly used punctuation marks (such as the period, comma, semi-colon, colon, parentheses, and others) are likewise used, but not necessarily with the meanings that the punctuation marks have in conventional English text. Other "special characters" (such as @, $, #, and %) found on many standard typewriter keyboards are relatively seldom found in a POL but are often replaced by symbols that are customarily used in the applications for which the POL is intended. The arithmetic symbols (particularly + and X) are frequently encountered with their usual meanings, but there are many others which have been designed to portray special functions.

NOTES

CHAPTER V

SIX STEPS OF PROGRAM DEVELOPMENT

In any complex discipline such as music, mathematics or science, there is a need for systematic development of steps which will lead to a desired result. The more complex the subject is, the greater this need will be.

In the development of a program for a computer, there are six basic steps that will provide the logical sequence necessary to produce an efficient and successful result. These six steps:

(1) Problem analysis

(2) Flowcharting

(3) Coding

(4) Debugging

(5) Production run

(6) Documentation

are defined and illustrated in the following paragraphs.

PROBLEM ANALYSTS AND PLANNING

1. INTRODUCTION

Before the actual writing of a computer program begins, the programmer must be concerned with planning the job.

"Programming" is a general term, and is used loosely. In its general meaning, it includes the analysis of the problem to be solved, the design of the solution and writing the instructions for the computer to perform the solution. These three separate areas of work are often referred to as "system analysis," "system design" and "coding."

We have separated these general areas of work into two major categories:

-System analysis and design
-Programming

In other words, we are using the word "Programming" in its narrower definition; that of instructing, or coding, the computer.

The fundamental function common to both areas of work; system analysis and design, and programming; is analysis. The means of conveying the results of analysis work is the flowchart. There are two types of flowcharts, system and program. The system flowchart is used to describe the work performed in system analysis and system design. The program flowchart is used to describe the work of the programmer.

It is the purpose of this unit to consider the nature and importance of analysis and to learn the meaning and use of the symbols employed in system and program flowcharts.

2. APPROACH TO PROBLEM SOLUTION

The approach to the solution of a problem utilizing an electronic computer is the same as the approach to the solution of any problem. This approach is:

a. Define the requirements.
b. Formulate a general solution.
c. Identify the available capabilities.
d. Match the general solution to the capabilities and arrive at a specific solution; a detailed set of instructions written in a particular language.

You can think of system analysis and system design being primarily concerned with the first two functions listed above; and the programmer, or coder, being primarily concerned with the last two functions.

The term "analysis" can be defined as the division of a whole into its parts or elements. Another way to think of the above-described solution, as related to a computer problem, is that the system analysis-design function divides the whole problem into fairly large parts; the programming function starts with each large part and divides it into parts small enough (instructions) for the computer.

Since analysis is creative in nature, it is most difficult to teach an individual how to analyze. Our approach to teaching the subject is to examine its general nature, provide guidelines, and teach by example. There will be several examples of analysis in this unit. There are many additional examples of analysis throughout the remainder of this course.

3. IMPORTANCE OF ANALYSIS

For the solution of a problem of any magnitude it is necessary to plan ahead carefully, taking into account as many of the facts and contingencies associated with the problem as possible. This is true whether the problem is adding a room to your house, setting up a new business, or finding the solution of a complex mathematical or business problem. Determining what facts are known and what results are desired, is another way to define "analysis."

In the case of adding a room to the house, failure to do a good analysis will result in the work taking longer than expected, and costing much more than expected. Failure to analyze and plan can cause extra trips to the store for forgotten nails, fixtures, paint, etc. This could easily increase costs 10% or more. Similar oversights could raise the cost (and hence decrease the profit) even more. In this example, the penalties are additional cost, frustration and inconvenience; the room will still get built.

In the example of going into business, a similar situation exists. However, consequences of inadequate analysis are more catastrophic since, in general, the business will fail and the money invested will be lost.

In the analysis of a mathematical problem, the solution may be beyond reach until the problem is carefully analyzed and broken into small sub-problems, each of which is small enough so that each portion can be solved easily. If a computer is to be used, the problem must be broken into even smaller pieces, since the basic language of a computer is extremely limited.

4. ANALYSIS OF A SIMPLE PROBLEM

To illustrate the elements of a problem analysis, let us take a simple non-mathematical situation. Suppose you were asked to plan and conduct a group picnic. Before ordering the food or doing a number of other tasks, you, as a prudent individual would analyze the situation. Your analysis might appear as is shown here:

Analyze the preparation for a group picnic.

Solution:

1. Determine three possible dates when the picnic grounds are available.
2. Take a random sample of the potential attendees to see which date is preferred.
3. Reserve the picnic grounds for the chosen date.
4. Prepare a questionnaire for those who are eligible to attend to determine if they are planning to attend.
5. For those who are planning to attend, determine if they prefer hot-dogs at $1.00 per person or hamburgers at $1.50 per person.
6. Tally the returned ballots.
7. Compare indicated results with the recorded results of last year's picnic.
8. If there is reasonable agreement, order the food.
9. If there is little or no correlation, take a repeat poll from a random sample and try to determine why the results were not as expected.
10. On the day before the picnic, call the weather bureau to determine if the day is expected to be warm or cool.
11. If the day is to be cool, order 80 percent coffee and 20 percent cold drinks. If the day is to be warm, order 30 percent coffee and 70 percent cold drinks.
12. On the day of the picnic, at 8:00 A.M., call the caterer to verify delivery and to give the last minute count of attendees.
13. After the caterer arrives verify that he has brought everything required.

This highly simplified example enables us to make a number of observations and to draw a number of conclusions.

a. An analysis must yield a set of indicated actions that occur in chronological order. For example, in the case of the picnic, we would not order the food before determining the amount or kind to be ordered, and the date on which it was required.

b. Contingencies must be provided for. In the example, steps 8, 9, and 11 provided alternate actions depending on the situation occurring at that time.

c. It is highly improbable that any two people would analyze a given complex situation the same way. If ten different people were asked to make the picnic analysis, ten different analyses would have resulted, some better than others, even though all would yield the correct final result.

Let us consider each of these points a little further.

The fact that the events in an analysis must be chronological serves a most important function. In a complex manufacturing situation it permits planning of ordering and manufacturing so that inventory is minimized, "slack time" is efficiently used, and no function is delayed because of the failure of some key component to be available when it is needed. In fact this particular kind of analysis has been highly developed using computers, and has been given the name PERT. (The letters stand for Performance Evaluation and Review Technique.)

As far as contingencies are concerned, in the case of the picnic it would not be catastrophic to fail to consider all possible alternatives that might occur. Human beings, being able to think, can handle unexpected events as they occur; although the "on-the-spot" solution would probably not be as efficient as if prior planning had been done. In the case of a computer, as we shall study later on, failure to provide instructions telling the machine what to do when a contingency is reached will usually stop the computer. A computer by itself, cannot figure out what to do. It does not possess reasoning capabilities.

5. CREATIVE NATURE OF ANALYSIS

The reason for specifically pointing out that the analysis of a situation is not unique, is to relieve the beginner in data processing from the worry that his analysis will not be correct. Analyses are as individual as the art forms produced by different artists, or as music produced by different composers.

An analysis is correct if it produces the correct end results and provides for all contingencies along the way. A correct analysis will be a <u>good</u> analysis if it is efficient; in the time spent making it; in the time necessary to automate its solution on a computer; and in the cost of the solution on the computer.

6. GUIDELINES FOR ANALYSIS

There are several guidelines for performance of analysis work. The guidelines involved are easy to state, but difficult to follow in actual practice.

One of the first guidelines is to *define* the problem to be analyzed. Where does the problem start? Where does the problem end? What is the cause of the problem? Should the problem be solved, or would it be a better course of action to live with the problem? Is solution of the problem within the realm of possibility? Will the cost of solution be reasonable?

No single individual can solve all of the problems in the world. Likewise, there is usually no single individual who can even solve all of the problems involved in a given organization. Accordingly, the first step usually results in narrowing the boundaries of a problem in order that it can be analyzed and solved. This matter of problem definition is so important that many large organizations establish it as a completely separate phase in the total solution of a problem. For example, all large U. S. Air Force programs have a "PDP" phase - a "Problem Definition Phase."

The second guideline is to be *inquiring*. *Make no assumptions.* Investigate all of the possibilities. Insist on facts. Face the facts. Attempt to physically look at the problem. Examine documents. Talk to the people involved in the problem. Keep in mind that people involved in a problem often are too close to it, or may not want to admit the facts.

The third guideline is to *quantify* the problem. State the problem in terms of quantities. There is a vast difference between the abstract concept and the realities. In concept there is no difference between 100 and 1,000,000. Keep in mind we are not analyzing problems in order to talk about them--we are analyzing problems in order to solve them.

The most common mistake made in the design of data processing systems is to start at the wrong place. As mentioned earlier, the one and only place to start the design of a data processing system is with the results desired. The desired results will lead to the establishment of the operating records and reports required. Once you have determined the operating record and report requirements, you can then determine the input data that needs to be recorded.

Another general guideline with reference to system analysis and design is to know your data processing aids. This includes not only computing equipment but all types of data processing and auxiliary equipment. Also, you should have a working knowledge of communication circuits and communications equipment. Equipment is constantly being improved.

There are literally thousands of design engineers concerned with inventing more capable, more reliable, less expensive equipment to aid in the processing of data. It can pay you well to be certain you are knowledgeable of the tools that can be brought to bear on a problem.

The final guideline with reference to systems analysis and design is to search for similar problems and observe how they were solved. Do not try to re-invent the wheel. Use the experience of others to your advantage. You will find that the individuals concerned with systems are more than willing to share with you their successes, and their failures; particularly if you pay them the compliment of asking for their assistance.

In program flowcharting, the best specific guideline is to pay very close attention to your "decision" operations. The decision operations change, or can change, the direction of the flow of work being performed by the computer. Probably the most common mistake made by programmers is to fail to account for all of the actions that can take place in a decision operation; and further, not to account for the chain of actions that can take place between different decision operations. This is the mistake most often responsible for the "looping" of a computer.

Another common mistake in program flowcharting is to fail to analyze the possible results of arithmetic operations. It is time well spent to check on the largest size a number can become during successive arithmetic operations, and to check whether a number can become negative.

Finally, in both system and program flowcharting work, take your time. Do not rush. Check your work. Be on guard for foolish mistakes. Analysis work, like all other creative work, takes time.

7. MODELS

Some problems are so large and complex that normal analysis techniques are inadequate. In these cases, one of the techniques employed to effect an adequate analysis of the problem is to build a "model." By analysis of the action of the model under different stimuli the actual problem is "simulated," and a better understanding is gained of the real problem.

The term "model" usually has an adjective associated with it, such as "Physical Model," "Mathematical Model" or "Numerical Model."

As the name implies, physical models are usually a physically scaled down version of the actual.

Mathematical models are normally utilized when the problem is so large that a physical model cannot be built; or it is uneconomic to build a physical model. In mathematical models, the various physical characteristics of the actual problem are represented by numbers and equations which are manipulated to represent the actual problem. This rather advanced area of work is known as "operations research," and is not one of the subjects of this course.

C. FLOW CHARTING AND DOCUMENTATION

1. INTRODUCTION

A pictorial representation usually carries a given amount of information to the reader in the shortest possible time. For this reason, system analysts and programmers have adopted many different pictorial representations, or symbols, as aids in describing their work.

The purpose of this unit is to consider the flowcharting conventions and flowcharting symbols used by the programmer. Paragraphs 2 through 7 cover program flowcharting; paragraphs 8 through 14 system charts and standard symbols.

2. THE PROGRAM FLOWCHART

A program flowchart is a graphic representation of the operations to be performed in the processing of data, and is usually the first form of documentation used by the programmer. The level of detail is commensurate with the means being employed for the processing of the data. Due to the limited vocabulary of electronic computers, the level of detail for program flowcharts for computer processed data is very low. For this reason, computer program flowcharts are often referred to as "detail" flowcharts.

Program flowcharts are commonly used in both scientific and business data processing. The program flowchart usually is the last level of planning prior to machine coding.

The conventions used for writing program flowcharts, and the symbols used, will be developed by means of a set of examples.

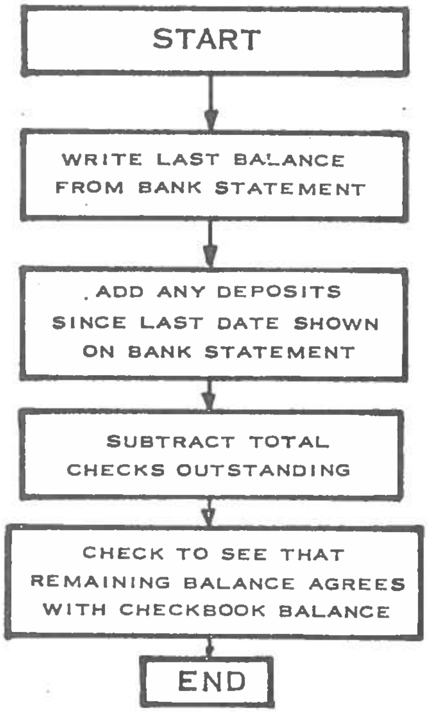
3. THE PROCESSING SYMBOL

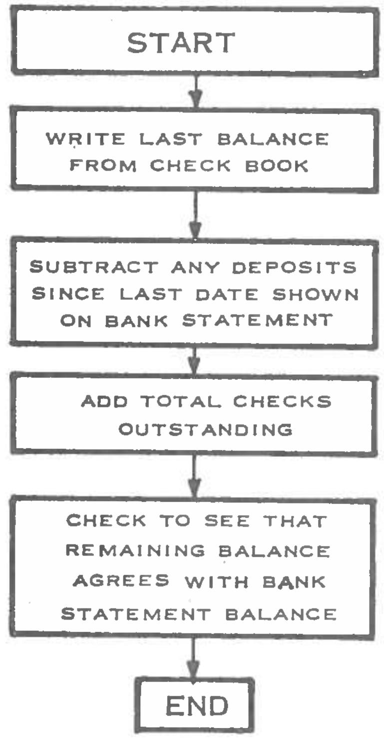
In an earlier unit we analyzed a simple problem. The result of the analysis was a set of difficult-to-read, and difficult-to-follow, instructions. We will now use the program flowchart technique for presenting the analysis of a similar, simple problem.

The only symbol we will use is the "processing" symbol. The processing symbol is a rectangle. Each separate processing operation (step) is written inside the rectangle, and the sequence of operations is indicated by directional lines:

> Example 1: Prepare a program flow diagram giving the procedure for reconciling one's bank statement at the end of the month.

ALTERNATE SOLUTION



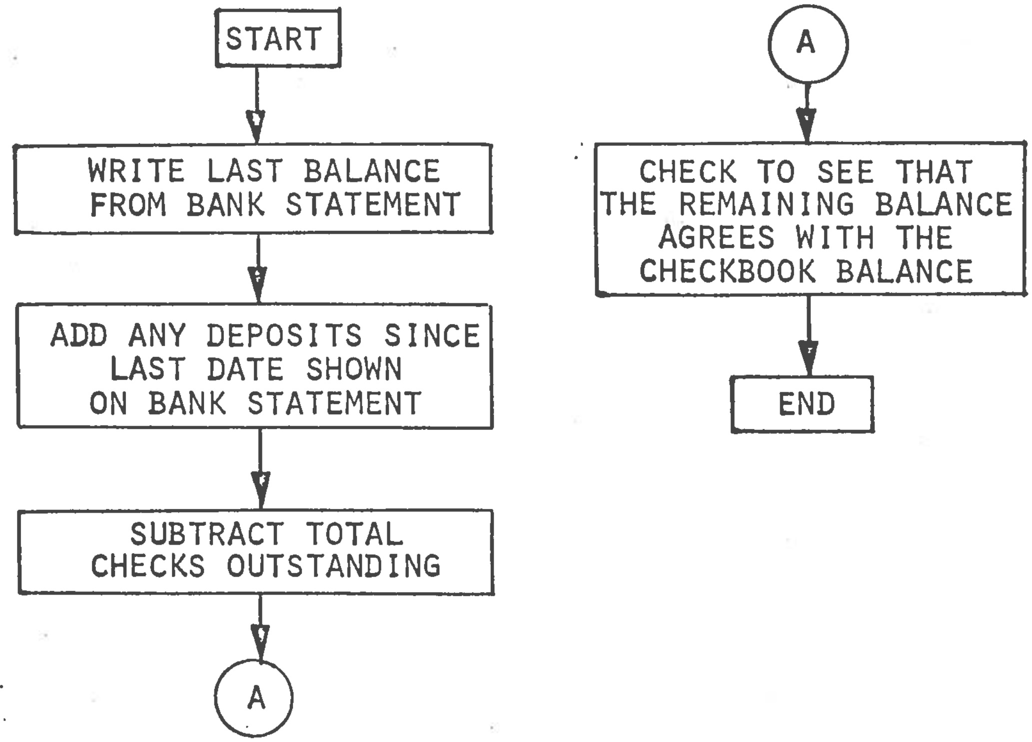


It should be noted from Example 1 that the relative locations of the rectangles--or blocks as they are often called--are not important. You need only follow the arrows from "Start" to "End" and the rectangles will be encountered in the correct order. The size of the rectangles is not important, although to make a nice appearing chart, they are ordinarily drawn the same size.

4. CONTINUATIONS SYMBOL

In large program flow diagrams, it may become necessary to continue the flow to another place on a page. The convention which allows this is the continuation symbol. It is a small circle containing a reference number or symbol. Continuation symbols are connected by a line. Another solution to the first example using the continuation symbol, is shown here.

Example 2: Second Solution to Reconciling Bank Statement



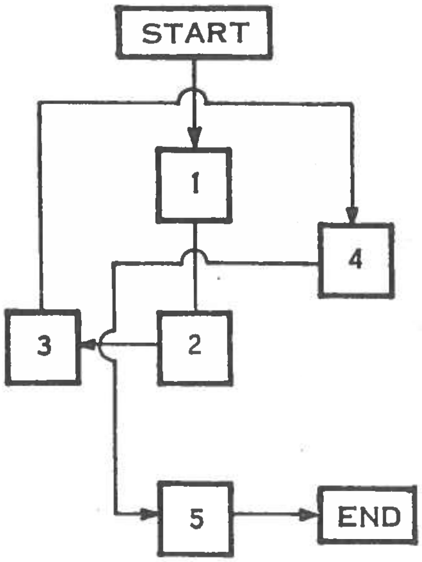
Another use of the continuation symbol is to permit drawing a block diagram without crossing any lines. This is shown in Example 3 where the numbers in the blocks are provided to show the desired sequence.

Although the solution shown in Example 3b is correct, and does illustrate the use of the continuation symbol, it is not a well displayed block diagram. A better arrangement is shown in Example 4a. The best arrangement is shown in Example 4b, provided there is sufficient available space.

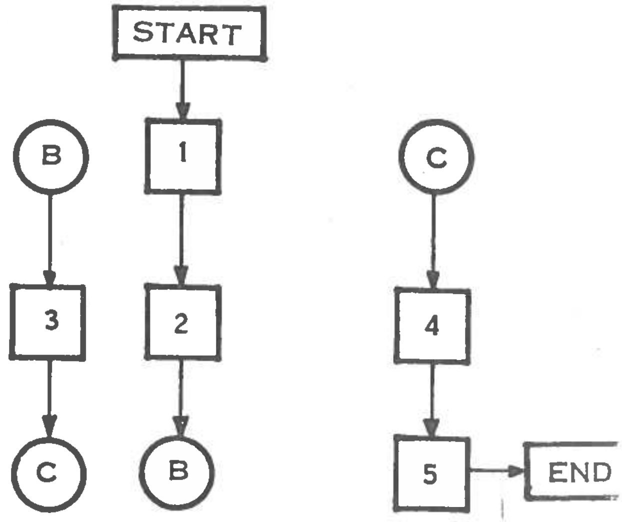
Example 3: Draw two block diagrams; one using crossed lines, one using connection symbols.

Solution:

a. Poor Technique



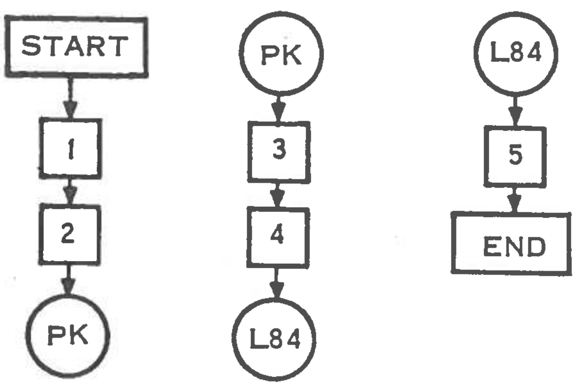
b. Better Technique



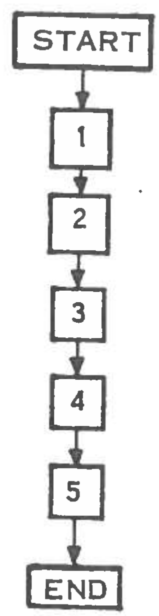
Example 4: Improve the arrangement of the block diagrams shown in Example 3.

Solution:

a. Good Technique



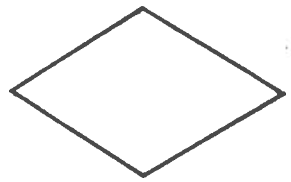
b. Best Technique



The best layout for a program flowchart is the simplest configuration.

5. DECISION SYMBOL

Recall that in the example of the group picnic analysis decisions had to be made on the basis of certain conditions existing at a certain time. In a program flowchart, a special symbol is used where a decision is to be made. The symbol is a diamond such as this:



Just as a statement block rectangle contains a statement, a decision diamond contains a question. Figure 3-1 illustrates decision diamonds and their appearance.

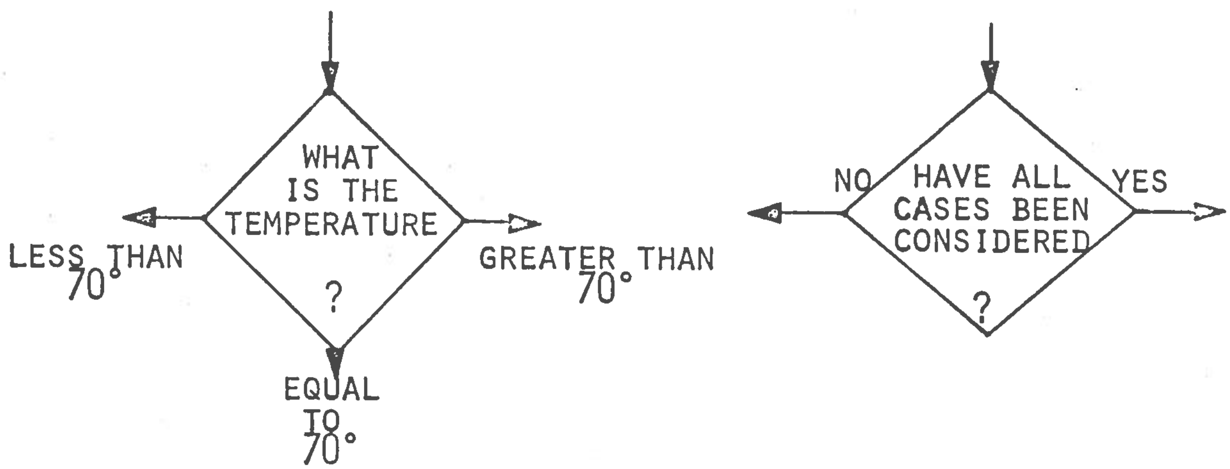


Figure 3-1 Examples of Decision Diamonds

In the case of a decision diamond, there is always more than one output for one input; thus the decision diamond creates a condition called "branching."

A classic example of the use of decision blocks is illustrated in the humorous analysis of starting to work in the morning. The analysis is shown in Figure 3-2.

The decision symbol is fundamentally concerned with "equality" or "inequality." The equality and inequality symbols are often used in conjunction with the decision diamond, to indicate the different courses of action to be taken as a result of a decision. The equality and inequality symbols are shown in Figure 3-3.

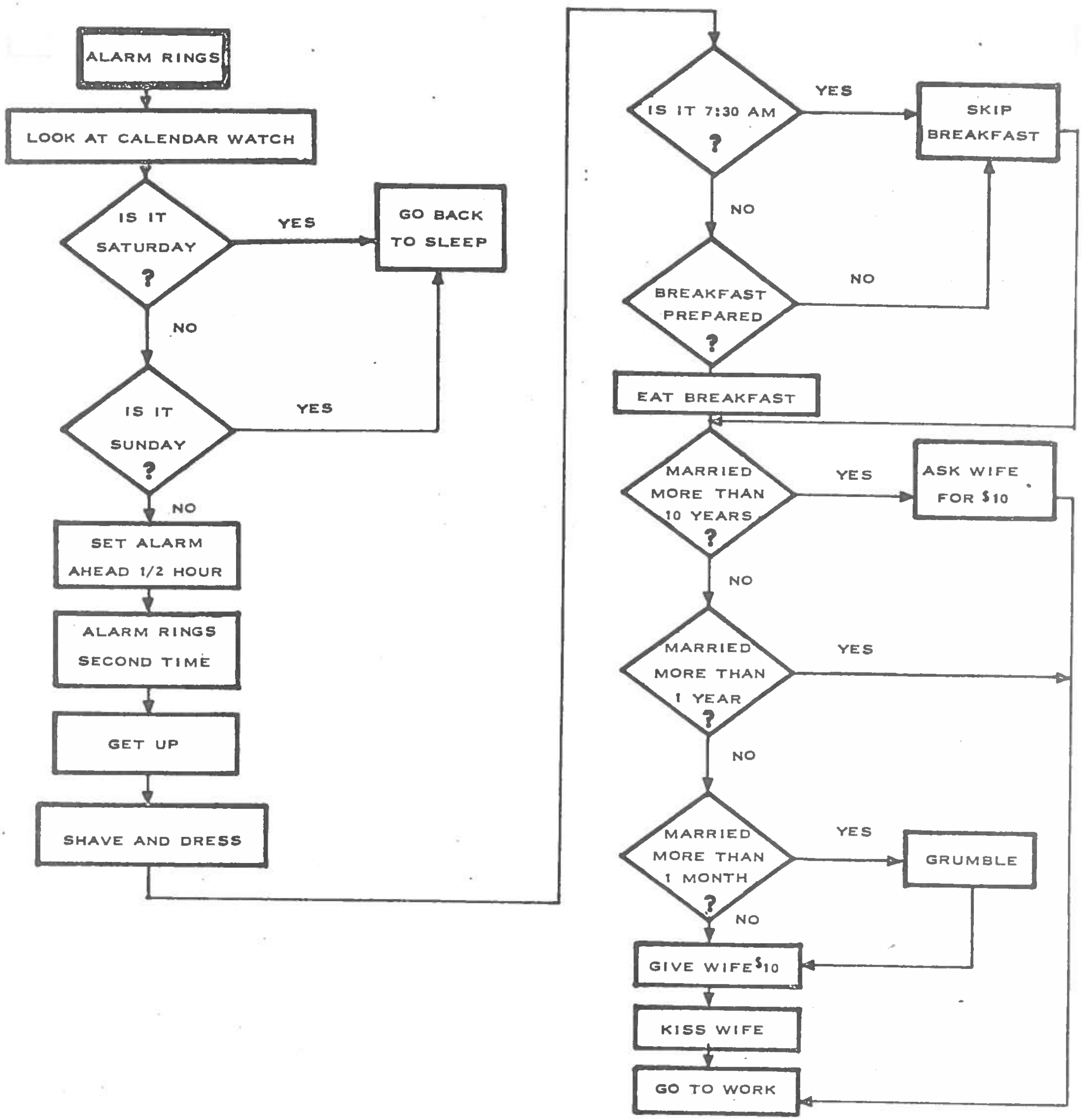


Figure 3-2 Program Flow Diagram Showing How to Start to Work

$=$ EQUAL

$>$ IS GREATER THAN

$<$ IS LESS THAN

$\neq$ IS NOT EQUAL TO

$\nless$ IS NOT LESS THAN

$\ngtr$ IS NOT GREATER THAN

$\geqslant$ IS GREATER THAN OR EQUAL TO

$\leqslant$ IS LESS THAN OF EQUAL TO

Figure 3-3 Equality and Inequality Symbols

When dealing with mathematical analyses, a "shorthand" symbolism is used in the decision diamond. In this notation, the question is reduced to two expressions separated by a colon. For example, a:b means "Compare a with b" or "how does "a" compare with "b"? Figure 3-4 illustrates how much simpler a decision diamond is when this notation is used.

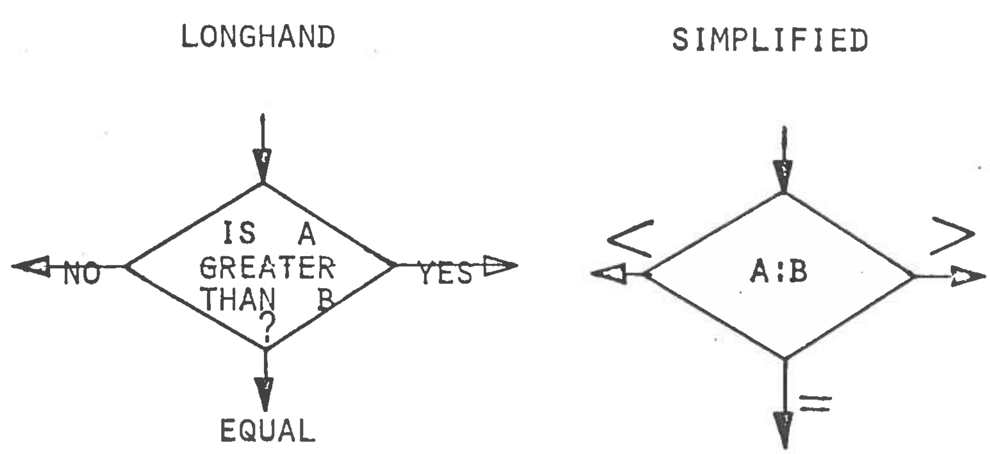
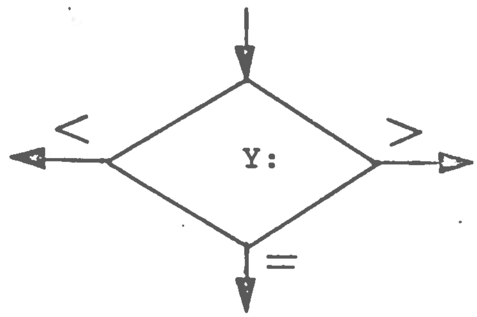


Figure 3-4 Example of Simplified Decision Diamond

If the decision is to determine if a quantity is positive, negative, or zero, the statement or symbols in the diamond are further simplified. Instead of writing y:0, the diamond may contain only y: as shown in Figure 3-5 (a).

To avoid any possible misunderstanding or confusion the positive, negative, zero decision diamond is often written as shown in Figure 3-5 (b).

(a) "Zero" Decision Diamond



(b) Alternate "Zero" Decision Diamond

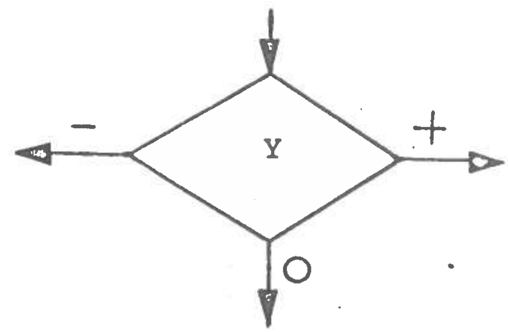


Figure 3-5 Zero Decision Diamonds

6. AUXILIARY SYMBOLS

The three symbols discussed above, namely the rectangle, the continuation symbol and the decision symbol, are the only three symbols actually required to produce a program flowchart for any problem.

However, as a means of providing better readability of the flowchart, six additional symbols have been defined, and are in common use.

Figure 3-6 shows all of the recommended program flowchart symbols and their representation. You will note by examination of the symbols in Figure 3-6 that the input/output, program modification, predefined process and terminal symbols are auxiliary symbols to the process symbol. These auxiliary symbols aid in readability by indicating, by the shape of the symbol, the nature of the processing being performed.

The offpage connector symbol is auxiliary to the connector symbol--its shape indicating the matching connection will be located on another page.

7. TYPICAL PROGRAM FLOWCHART

A typical program flowchart is shown in Figure 3-7. The flowchart assumes the card edit sub-routine has been previously

charted. The processing operation shown is that of reading four 80 column cards; performing an edit; writing a 320 character block to magnetic tape; testing for completion; repeating 2,000 times; and stopping upon completion.

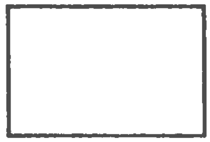
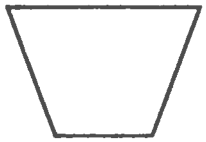
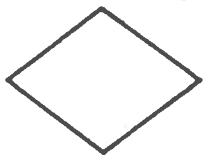
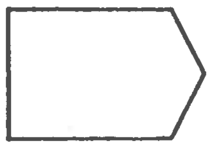
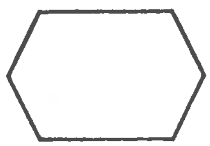
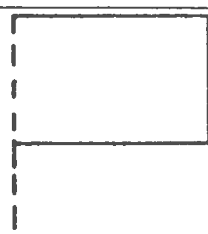
SYMBOL	REPRESENTS
	PROCESSING A GROUP OF PROGRAM INSTRUCTIONS WHICH PERFORM A PROCESSING FUNCTION OF THE PROGRAM.
	INPUT/OUTPUT ANY FUNCTION OF AN INPUT/OUTPUT DEVICE (MAKING INFORMATION AVAILABLE FOR PROCESSING, RECORDING PROCESSING INFORMATION, TAPE POSITIONING, ETC.)
	DECISION THE DECISION FUNCTION USED TO DOCUMENT POINTS IN THE PROGRAM WHERE A BRANCH TO ALTERNATE PATHS IS POSSIBLE BASED UPON VARIABLE CONDITIONS
	PROGRAM MODIFICATION AN INSTRUCTION OR GROUP OF INSTRUCTIONS WHICH CHANGES THE PROGRAM.
	PREDEFINED PROCESS A GROUP OF OPERATIONS NOT DETAILED IN THE PARTICULAR SET OF FLOW CHARTS.
	TERMINAL THE BEGINNING, END, OR POINT OF INTERRUPTION IN A PROGRAM
	CONNECTOR AN ENTRY FROM, OR AN EXIT TO, ANOTHER PART OF THE PROGRAM FLOWCHART.
	OFFPAGE CONNECTOR A CONNECTOR USED INSTEAD OF THE CONNECTOR SYMBOL TO DESIGNATE ENTRY TO OR EXIT FROM PAGE
	FLOW DIRECTION THE DIRECTION OF PROCESSING OR DATA FLOW
SUPPLEMENTRY SYMBOL FOR SYSTEM AND PROGRAM FLOWCHARTS	
	ANNOTATION THE ADDITION OF DESCRIPTIVE COMMENTS OR EXPLANATORY NOTES AS CLARIFICATION.

Figure 7-6 Standard Program Flowchart Symbols

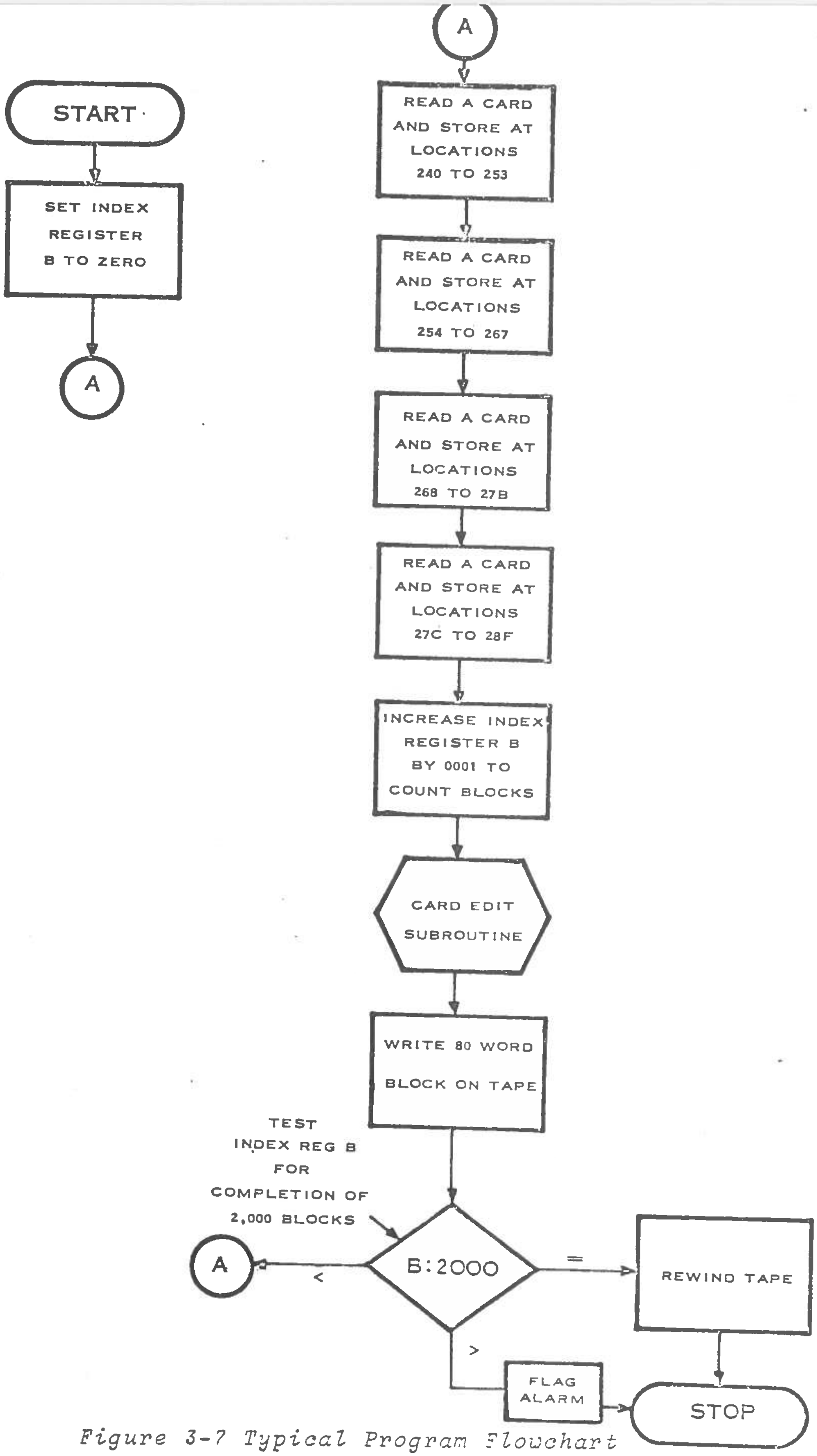


Figure 3-7 Typical Program Flowchart

8. SYSTEM FLOWCHARTING AND STANDARD SYMBOLS

Whereas the program flowchart is concerned with the detail description of the operations to be performed on data, the system flowchart is concerned with the total system. The program flowchart shows <u>how</u> the work is to be performed. The system flowchart shows <u>what</u> is to be accomplished.

System flowcharts are primarily concerned with source documents, reference files and reports. They are concerned with the physical form of source documents (paper document, punched card, punched tape, etc.), the physical form of file storage (magnetic tape, magnetic drum, magnetic disc, etc.), and the physical form of the output (printed report, cathode ray tube display, punched card, etc.). To provide the latitude necessary to indicate so wide a range of input, storage and output elements, a much larger symbol set is required for system flowcharting than is required for program flowcharting.

9. STANDARD SYSTEM FLOWCHARTING SYMBOLS

In general, the symbols used in system flowcharting represent, as nearly as possible, an outline of the device being referenced. Figure 3-8 shows the set of system flowchart symbols currently recommended as standard.

In many instances, the standard system flowcharting symbols are embellished to carry more specific meaning to the reader. Some of the embellishments typically used are shown in Figure 3-9.

10. DIRECTION OF FLOW

The basic element of the system flowchart is the line and arrow indicating the direction of flow. Ordinarily solid lines are used to indicate flow. However, in complex system flow diagrams, it is desirable to be able to differentiate between the physical movement of documents, or data, and the mere transfer of information. Solid lines are used for the first and dashed lines for the second.

For example, dotted lines are used to illustrate accounting and system control functions. Figure 3-10 shows the use of control lines in a simple business Data Processing application.

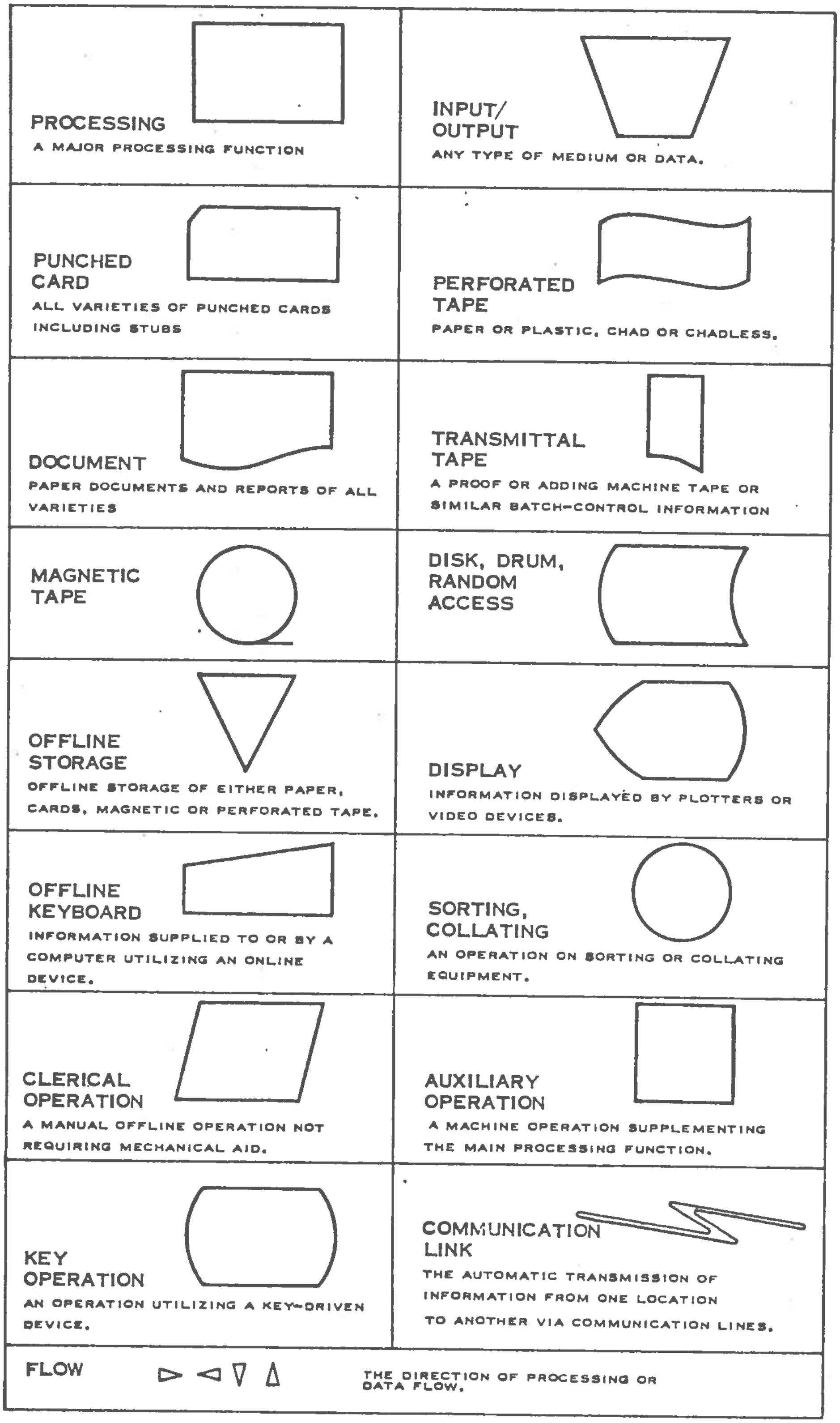


Figure 7-8 System Flowcharting Symbols

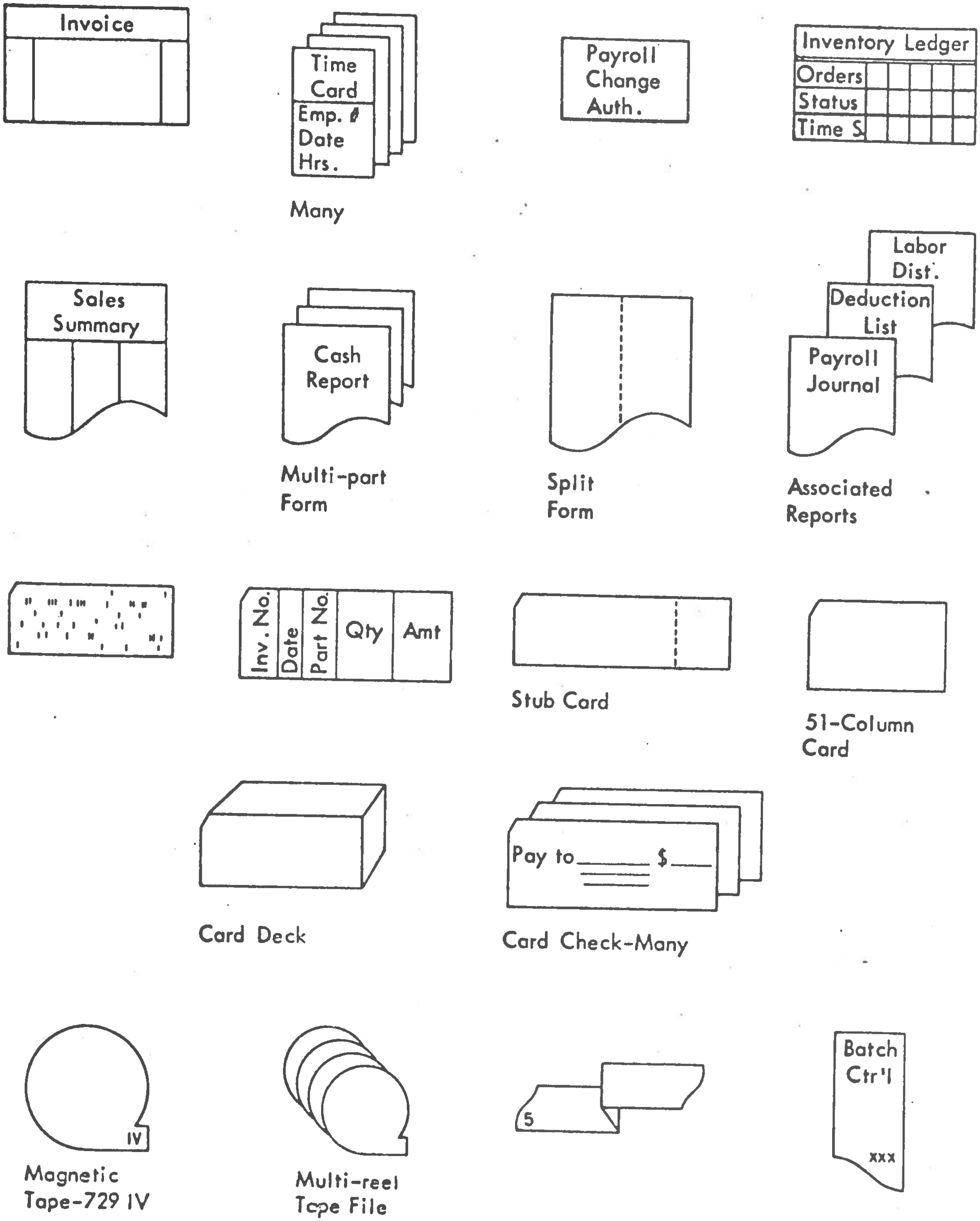


Figure 7-9 Variations of Standard System Flowcharting Symbols

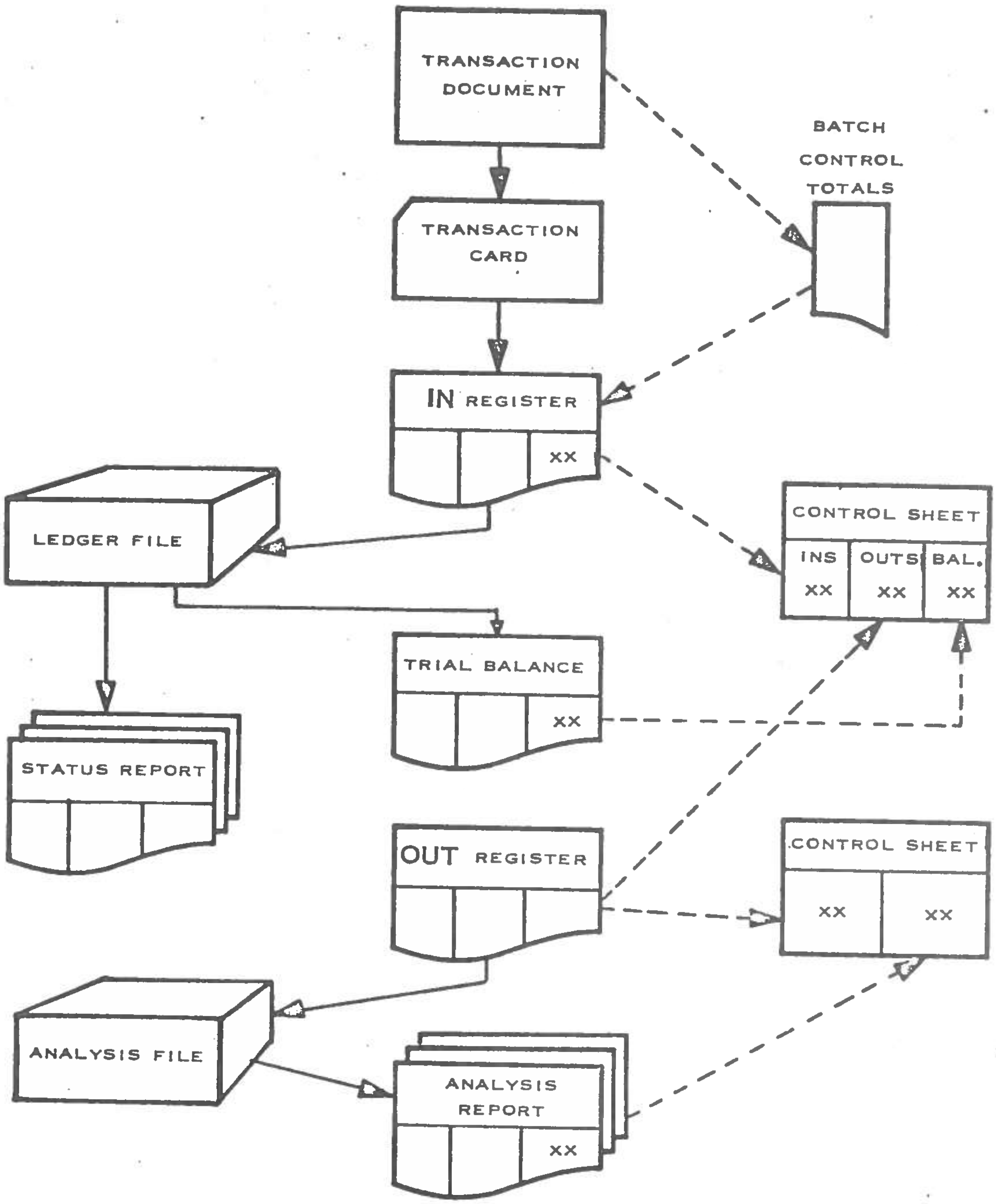


Figure 3-10 Simple Business Data Processing Application

11. SYSTEM FLOWCHARTING

A system flowchart is shown in Figure 3-11. Essentially, the system flowchart is concerned with data input, output and major functions.

As a means of comparison, refer to the program flowchart in Figure 3-7, which covers part of the work shown in the "Computer Run #1" processing function block shown in Figure 3-11.

Both flowcharts are concerned with the same general function. However, on one page, the system flowchart shows the type of documents to be processed; how machine intelligible input is to be obtained; the major processing functions to be accomplished in the computer run; the source of data for the control function; and the disposition of the documents.

In a similar size program flowchart, Figure 3-7, only details of the "card read" and "tape write" functions are shown.

Regarding the two sets of symbols, the only common symbols between the program flowchart and the system flowchart are the "processing" symbol and the "connectors." The connectors are utilized in the same manner in both types of charts. However, the rectangular processing symbol, in the system flowchart, indicates a major processing function; whereas in the program flowchart it indicates a group of computer program instructions.

12. STANDARD VERSUS NON-STANDARD FLOWCHARTING SYMBOLS

The flowcharting symbols shown in Figure 3-6 and 3-8 follow the recommendations of the American Standards Association, an organization supported primarily by the Business Equipment Manufacturer's Association. These are the symbols we recommend you use, and they are the symbols expected to be used in questions, solving problems and in examination answers.

However, we want to caution you that these flowcharting symbols, like many other aspects of electronic data processing, are not used exclusively by all computer personnel. You will encounter many deviations in flowchart symbols, particularly in flowcharts prepared prior to 1960.

However, if you are thoroughly familiar with the standards given here, and with flowcharting techniques, you should experience no difficulty in understanding any flowchart, regardless of the deviations from the standard symbols.

13. IMPORTANCE OF FLOWCHARTING

Many experienced programmers attempt to write computer programs without having first completely analyzed the problem and without drawing system and program flowcharts. The result is a poor quality program, which is difficult to debug, and which must still be documented with flowcharts at a later date.

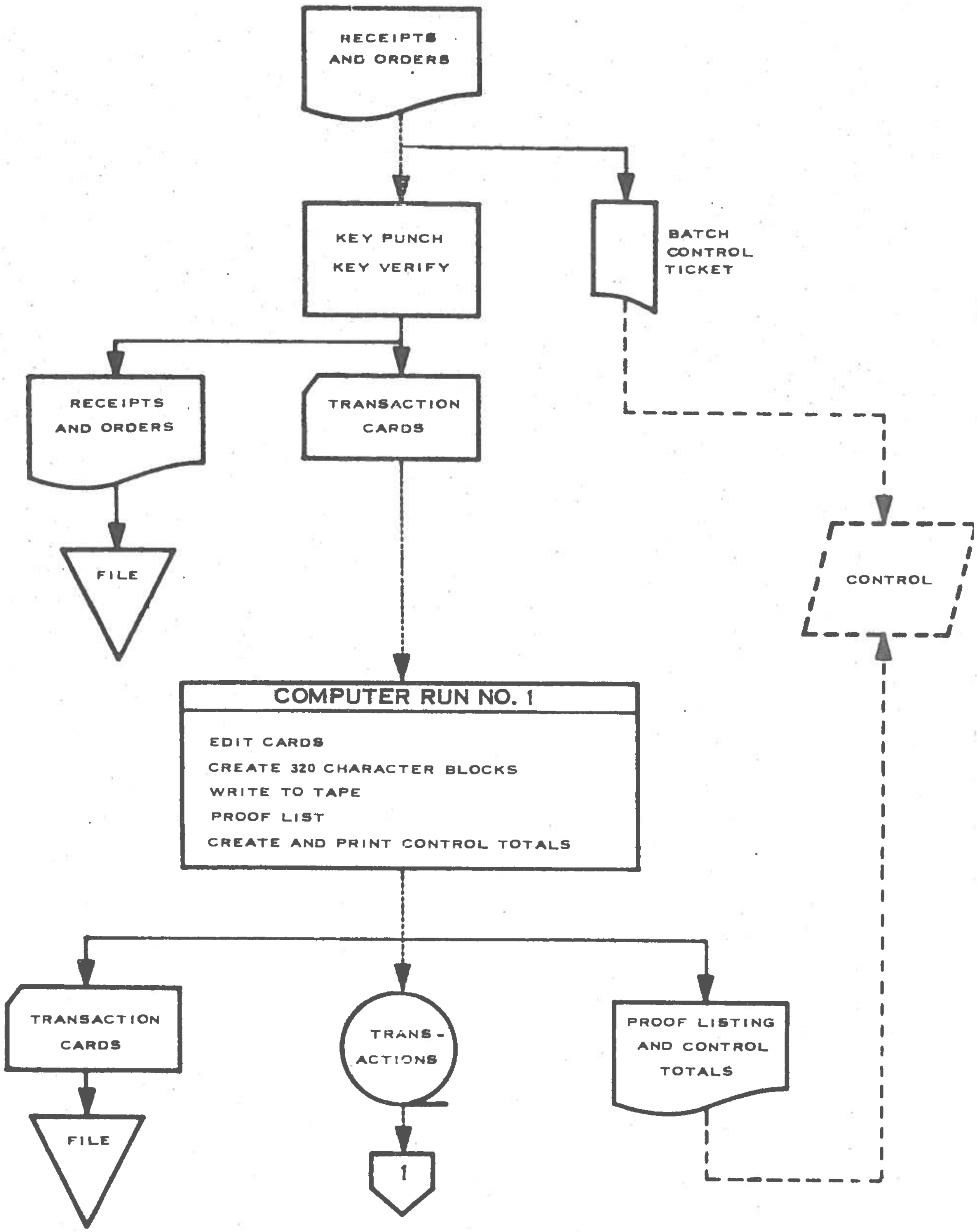


Figure 7-11 System Flowchart for Card to Tape Conversion

The situation is analogous to building a house without prior planning and then drawing up a set of plans to indicate what was built. Such a procedure would be costly and inefficient; the house would tend to be disorganized; and there is a good chance that when the house was nearly complete it would be discovered that the furnace or bathtub had not been installed and would not go through the door. It is like the classic story of a man building a boat in his basement only to realize when it was completed that he could not get it out.

By all means develop the good habit of carefully analyzing each job you do and of drawing all of the necessary flowcharts before you attempt to write a computer program.

Should you become concerned with the management of a computer project, you should insist that both system and program flowcharts be prepared covering all work being performed by the systems and programming staff. One of the prime benefits of flowcharts, in addition to their aid in analysis, is their aid in communication. Flowcharts provide an excellent means for management to be able to follow and control the activity of systems and programming personnel. They also represent an excellent means of communicating with the individuals outside the data processing organization for whom you are performing work. Finally, flowcharts are useful as a historical record of what has been done; and as documentary support for operating programs.

14. SUMMARY

Before a computer program is prepared, it is necessary that the method of solution and the details of the computer program be carefully analyzed, taking into account all possible contingencies and endeavoring to perform the entire operation as efficiently as possible.

The results of the analysis are shown pictorially using flowcharts made of symbols. Although some analysts and programmers write computer programs without drawing a system or programming flowchart, it marks them as amateurs. Invariably, their programs suffer in quality from lack of thorough analysis and planning.

Professional system and programming personnel recognize the value of careful planning and, of course, always prepare neat and accurate flowcharts before starting computer coding.

15. CODING is the translation of problem solving logic of the flowchart into orders or instructions for the computer. Thus far we have merely drawn pictures of the logic needed to solve the problem. Obviously, the computer cannot understand or interpret these graphic pictures. We must now translate our logic into an ordered series of instructions which, when executed, will produce the desired result.

Let us consider the problem of transcribing data from punched cards to a printed page. In the previous section we made a flowchart of the logic needed to achieve this goal. If we were to make an orderly list of the operations to be performed, it would look something like this:

a. If the last card has been processed, go to Step 6; if not, go to Step 2.

b. Read a card.

c. Does card column 80 contain a "P"? If yes, go to Step 4; if no, go to Step 1.

d. Print out card information.

e. Go to Step 1.

f. Halt.

Notice that the statements or instructions would be performed one after the other, or sequentially, except at three points. One of these points is in Step 5. Step 5 will automatically take us back to Step 1(this transfer is called a "branch"), and we will not proceed in sequence to Step 6. This type of order is called an "unconditional branch" since it takes us back to Step 1 regardless of any conditions.

Another point where our sequence may be broken is at Step 1. If the last card has been processed, the sequence of instructions will be broken, and we will go to Step 6. If there are cards remaining to be processed, we will continue sequentially to the next step. This is what is referred to as a "conditional branch" since the branch occurs if and only if the condition being tested for is found. What kind of branch do we have in Step 3?

The computer is capable of performing the above operations in the proper sequence if these statements or instructions are written in a "language" the machine can interpret. Once we have coded the sequence of operations into the machine's language, we have written a computer program. This program can be stored in

the machine (separate from any data) to process data in any way or order we so describe. The computer will then process the given data per the coded instructions one by one, sequentially, until an instruction orders it to break this sequence, either conditionally or unconditionally. These two concepts, the stored program concept and the ability to alter the sequence of operations to be performed (as determined by conditions of the data) give computers their tremendous speed and versatility.

Now let us take our series of instructions and put them into a form (by coding them) that the computer is capable of interpreting. The only language that the computer is capable of "understanding" is called the <u>actual</u> or <u>absolute</u> language. However, other "higher level" languages do exist; we will discuss them later.

Since the coding to be used will in most cases depend on the type of computer being programmed, we will use the IBM 1401 data processor as an example in this section. If we wrote the above program in the absolute language of the IBM 1401, it would look like this:

```
B 365 A
1
B 352 080 P
B 333 b
M 080 280
2
B 333 b
.
```

Writing instructions in an absolute language becomes very difficult and tedious as the complexity of the problem increases. Higher level languages (symbolic languages) were developed to make this process easier and faster. Because the machine cannot interpret these languages, they must be translated to their machine language equivalent (assembled or compiled) before the computer can perform a given task. The translation is performed by the computer using existing programs specifically written to handle a particular symbolic language.

Autocoder is the symbolic language written for the IBM 1401. Since we are using the IBM 1401 as our example computer, let us look at our problem solving statements written in that language.

```
START  BLC  END
       R
       BCE  HERE 080 P
       B    START
HERE   MLC  080 280
       W
       B    START
END    H
```

Since the menemonics (abbreviations) used in the above example are closer to our English language, this language is easy to use and remember for the programmer. There are in existence and use a still higher level of symbolic languages.

This step in the development of a computer program can be a time consuming and tedious task and should only be taken after a great deal of preparation. The validity and worth of the coded computer program depends upon the quality and quantity of analysis preceeding processes such as the one described in the next section.

16. DEBUGGING is the correction of logical and typographical errors. Programming and coding involve so much detailed work that most programs do not operate correctly when first tried. Any sizable data processing application provides hundreds of opportunities to make mistakes that will completely invalidate the program. These may range from simple slips of the pencil that result in non-existent addresses, to block diagramming and Flow Charting errors that destroy the logic of the program, to misunderstandings of intended purposes. Thus, it is necessary to "debug" the program (locate and correct errors) and to test it with "test cases" to be sure that the program properly processes the data. This process of detecting and correcting errors and proving correctness of the program can easily take weeks. All of this goes under the name of "program checkout," which is commonly referred to as "debugging".

One practice, called "desk checking," that is strongly recommended, is to check all programs very thoroughly before they are assembled or compiled. This should be a character-by-character check to make sure every symbol and every operation code is correct, that 0's (numeric) and O's (alphabetic) are properly distinguished, etc. Time saved in desk checking will save much more time later.

After thorough desk checking, the program is then put into the machine with suitable test data to see whether it computes correct results. Usually, the machine stops before producing any results ... Some errors may be indicated on the computer console. When this occurs, it is fairly easy to get back to the source of the trouble, correct it, and try again.

A second aid is the "memory dump" or "memory printout". This provides a printed listing of exactly what was in core storage at the time the program was stopped.

Another important debugging technique is to check out the program in sections by providing for the printing of intermediate results. This allows the source of errors to be narrowed down. This brings up a point that should be kept in mind: debugging is such an important part of the whole process of computer program development that it must be planned for in the writing of the program in the first place. The provision of instructions to print intermediate results is only one example of this sort of planning.

The final check that should be made is called "pilot processing" or "volume testing". This of using large quantities of applicable data which has previously been processed either by machine or manually. The results are then checked against the previously accepted results.

Another possible cause of errors is the malfunctioning of the machine itself. Corrective action here involves actions such as cleaning contacts, checking for loose wiring, or replacing components.

Finally, if before you read this section you thought debugging was the extermination of nasty insects, it is hoped that you now realize that debugging is one of the most important cost and time saving steps in computer program development.

17. PRODUCTION RUN is the running of the program with real data and producing an end result. Although many months of time and hard effort may have been devoted to problem analysis, flow charting, coding and debugging, the actual production run may only take minutes. It is possible that the problem solved by the computer could have been solved manually in a shorter time if one considers the time devoted to the steps leading up to the production run. Why then should computers be used if this is true? The answer is simple. Once a program is proved to be reliable, it can be used over and over again for solving future problems which contain the same type of data. Also, it may be a simple matter to modify the existing program to solve similar but not identical problems.

Different uses of the results may require various forms of output or display. The most common method of output is to print the results on paper. New requirements may call for the output to be recorded on magnetic tape, transmitted by electromagnetic means, or perhaps displayed on a cathode ray tube. It is an easy matter to modify the existing program so that it will yield its results on a different medium. If this were done manually, the entire problem would probably have to be recalculated from beginning to end.

Although it appears that the execution of the production run is a simple matter, there are many factors which must be considered so that the production run will be done in the most efficient

and saving manner. These factors are mainly administrative nature and include:

1. Scheduling the usage of the computer so it will be of greatest advantage to your project.

2. Ensuring that your usage of the computer will not interfere with other projects.

3. Ensuring that all input data will be available at the time the production run is to start.

4. Providing for continuous inspection of the output as the production run progresses.

5. Ensuring expeditious delivery of the results to all interested parties.

As you can see, it requires considerable coordination of efforts to ensure efficient usage of the computer. Since it can cost hundreds of dollars per hour to operate the computer, a few minutes saved will save a considerable amount of money.

The production run may be the easiest step in computer program development, but it should be remembered that it is the ultimate goal, and all of the previous steps of computer program development exist only to reach that goal.

18. DOCUMENTATION is the "write-up" of all the initial, intermediate and final facts relative to the solution of a given problem. DOCUMENTATION, the final step in writing a computer program, is basically as important as the first, problem analysis. At this point in the program preparation the investigation of the problem and analysis is complete, a block diagram and flow chart have been written to set forth the logic needed to solve the problem, a set of coding has been written in program form following the logic of the flow chart and block diagram and the written program has been corrected and finalized through the debugging and testing processes. The program, as written, should now be in production.

A programmed solution to a problem is practically worthless unless the steps taken to reach this ultimate goal have been noted. For instance, of what value would a new miracle drug be if the thousands of tests over a long period of time needed to produce this drug were not documented fully? It would be a difficult task to retrace the steps to the solution. Logically then, the final step to be taken in the development of any computer program is to combine all the information derived and facts relative to the program in a document.

Documentation should make the analysis and solution of a problem available to all who are interested in this particular problem or related problems. Since a program is primarily an expression of the thinking or logic of one person or small group of persons, documentation should present enough information so that any programmer can follow the logic to make future modifications or changes to a program, if needed. Also, it should provide enough information for subsequent analysis by Data Processing Systems personnel to determine if the right machine and methods are being used to solve any given problem. It can serve as a management tool for evaluation of programming personnel and practices. Finally, documentation can provide the information needed for efficient production run of a program.

There are basically two types of documentation. They are the operating instructions and program write-up.

Although a program may actually run to completion in a few minutes or hours, a great deal of machine time can be wasted just setting the forces in action unless preparations have been made beforehand. For instance, input data must be gathered together and verified as being the correct data for a program run, tapes must be mounted on tape units, cards must be loaded into the card reader or punch, the printer must be checked, switches must be set on the console, the operator must continually monitor a program while in execution, output tapes must be labeled for identification and all output data must be gathered together for disbursement. To facilitate efficient production runs the operating instruction should generally include some or all of the following:

1. Job number

2. Name of programmer

3. Brief description of what program is doing

4. Date of completion of program and last modification

5. Flow chart

6. Block diagram

7. Instructions to operator for error procedures, tape labeling, descriptions of tapes, disposition of tapes, rerun instructions, average run time, switch settings, etc.

8. Examples and descriptions of parameter cards

9. An assembly listing of the program

10. Samples of input and output data

The program write-up, a more formal document, should be all conclusive, perhaps even duplicating some of the information found in the operating instructions. It should cover from the initial analytical studies of the problem down to conclusions and final results. Basically this documentation should include the following:

1. Brief description of the problem

2. Description of the solution or solutions

3. Description of the program as written, including possible modifications, coding that cannot be modified, explanations of amcro-subroutines or specialized subroutines that might be used in other programs, etc.

4. Assembly listing of the program

5. Samples of the input and output data

6. Descriptions of the input and output formats used, discussion of frequency of preparation of input and output data, etc.

7. Timing schedules for data submission and disposition

8. Handling of special conditions

We can then say that although we have presented documentation as the last step in developing a program, a programmer must keep it in mind throughout all the previous development stages.

NOTES

APPENDIX A

AN INTRODUCTION TO ALGOL

ALGOL is probably the most widely accepted POL for general mathematical problems. Many other POL's for the same types of problems are in use at various computer installations, and these other languages have varying degrees of similarity to ALGOL.

The symbols in ALGOL include the fifty-two lower and upper case letters of the alphabet, the ten decimal digits, most of the common punctuation marks (not all of which have meanings corresponding to their conventional meanings), and about a dozen special symbols, two which will be mentioned here. One of the special symbols $\uparrow$ is for exponentiation. The quantity preceding the symbol is the base and the quantity following the symbol is the exponent as illustrated by the following examples: $(x + y)\uparrow m$ means $(x + y)^m$; $x\uparrow(m\uparrow n)$ means x raised to the m^n power; and $x\uparrow y\uparrow z$ is the same as $(x\uparrow y)\uparrow z$ which has the meaning of x^y raised to the z power. The other special symbol to be mentioned is written on paper as the sequence of two symbols, the colon and the equality sign, but the meaning of this symbol cannot be explained clearly until after the concept of the identifier had been introduced.

In ALGOL, an identifier is any sequence (string) of letters and digits (no spaces) that begins with a letter. Examples of identifiers are x, xy, and jo2N. Note the xy is not the product of x and y, which would be written as x x y. Each identifier that may be used in a program has whatever meaning that may be assigned to it in that program. About twenty symbol sequences that would be identifiers within the definition just given are reserved for special purposes. These particular symbol sequences all form English language words that have meanings in ALGOL. The words are mnemonic in that their ALGOL meanings are similar to their conventional meanings. In recording a program in ALGOL, the words are always either underlined or written in boldface type, such as "comment", to assist a person reading the program, but the emphasis is of no consequence to the computer on to the translator used with ALGOL. The translator does, however, recognize these particular words and initiates appropriate response in each case. A program in ALGOL consists of a sequence of statements. An example of a simple statement would be:

$$jo2N := x + y\uparrow 2$$

This statement has the effect of indicating that y^2 should be added to x (x and y being identifiers that presumably have been given assigned meanings earlier in the program) and that the identifier jo2N is assigned to represent the result. In general, there is no limit to the length or complexity of the mathematical function that can be specified by a single statement.

Most commonly, successive statements are written on successive lines on a piece of paper, but this practice is for ease in understanding the program, and is not a requirement of the language. The lines may also be indented in various patterns to indicate statement groupings, but to a computer or translator, a program is just one long string of symbols. Semicolons are used to separate individual statements.

Statements may be labeled. Ordinarily only symbol strings that are identifiers are used as labels (but integers can also be used), and the fact that an identifier is a label is indicated by placing the identifier at the beginning of the statement and following it with a colon. For example, the previous statement could be labeled a^6 by writing the statement as follows:

a^6: jo2N := x + y ↑ 2

Among the reserved identifiers are a few which serve the purpose of specifying the character of the parameters represented by other identifiers, For example:

REAL x

is a statement, and it means that x is a variable that can assume any value within the capacity of the computer on which the program is to be operated. The statement INTEGER x has the meaning the x may assume only integral values. The statement BOOLEAN x means that x is a Boolean variable and will have only one to two possible values. The identifier ARRAY is used to define matrices, a discussion of which would go beyond present purposes. Actually, in the instances cited in this paragraph the identifiers and statements are called "declarators", and "declarations", respectively.

Jumping in an ALGOL program is accomplished by the "go to" identifier. The jumping can be unconditional, but as encountered in most programs it is combined with other programming functions in any of many different possible ways, one example of which is the following statement:

'IF' A ≤ X+Y 'THEN' 'GO TO' A6 'ELSE' G := X-Y

The statement means that if a is equal to or less that the sum of x and y, do nothing except "go to" the statement labeled a^6 to find the next function to be performed, but if "a" does not meet these conditions ("else"), subtract y from x and assign the identifier g to represent the difference. In the latter case, the next function to be performed. In preparing the machine language program, the translator must determine the instructions necessary for adding x and y, for comparing "a" with the sum, for conditionally jumping to the correct instruction in the set of instructions corresponding to statement a^6, for subtracting y from x, and for storing the difference at an address known to the program.

Statements in ALGOL can be grouped into what are called compound statements by beginning and ending the group with "BEGIN" and "END" respectively. A compound statement has the same status as any other statement in that it may be labeled in the same way and may, in turn, be combined with other statements to form "higher levels" of compound statements. Any identifiers that may be specified within a compound statement have their specified meanings only with respect to other statements, within the same compound statement. Therefore, any given identifier, notably a commonly used one such as "x," may be used with totally different meaning in other parts of the program.

Statements containing the reserved identifiers "for ... do ..." allow many types of problems to be written very simply in ALGOL. The "for" part of the statement specifies the conditions to be fulfilled when performing the "do" part of the statement. An example, which incidentally introduces two other special identifiers, is:

'FOR' X:=3 'STEP' Ø.1 'UNTIL' 4 'DO' Y[X] : X↑2

This statement has the meaning that x is initially given the value 3 and is increased (stepped) by increments of 0.1 until it reaches the value of 4, and for each value thus derived the square of X is computed and is designated y_x, which in ALGOL is written y[4], the brackets being used to signify a subscript. In general, the "do" part of such a statement can have the form of any other statement, including a compound statement, provided the net result "makes sense".

A person embarking on the task of preparing a program in ALGOL would want to know much more about the details of the language than have been presented here. ALGOL itself does not cover the input - output functions of a computer. Because of this and other factors, a programmer must be familiar with the specific "dialect" of ALGOL that may have been selected for a specific translator, which in turn was prepared to match the physical input-output characteristics of a specific computer.

Although ALGOL consists of countless different sets of statements that will accomplish any given end result, the efficiency of the machine language programming generated by the statement sets varies. Some such programs will be much more efficient than others. About the only thing that can be said about achieving machine language programs of high efficiency is that when this factor is important, the program must usually be written in machine language without the aid of ALGOL.

APPENDIX B

A SIMPLIFIED APPROACH TO FORTRAN

FORTRAN (an abbreviation for FORmula TRANSlation) is a language for which compilers are universally available. There are many variations in FORTRAN compilers, but most versions are sufficiently similar that minor modifications in any FORTRAN program will make it acceptable for translation by most FORTRAN compilers, and hence usuable on most computers.

A FORTRAN program consists of a number of statements, or instructions to the computer. To develope some fluency in FORTRAN, we will first examine some of the basic conventions governing FORTRAN programming.

There are a number of different kinds of statements that make up the list of possible FORTRAN instructions. The basic algebraic statement is one of these. It consists of a variable, an equal sign and some expression to the right of the equal sign. When such a statement is encountered the computer gives to the variable on the left side of the equal sign the value of the expression on the right side of the equal sign. Thus, upon encountering the statement:

```
AHC = 40.00
```

the computer stores in its "memory" the value 40.00 for AHC. Later, if we call for the value of the variable AHC to be pointed out for us, the value 40.00 will be pointed out (assuming that the value of AHC has not been changed through subsequent operation).

The expression to the right of the equal sign can be a more involved mathematical expression; for example: AHC = 70.00 + 20.00 - 50.00. The computer will perform whatever computation is called for by the expression to the right of the equals sign, and then will give to the variable on the left of the equal sign, the value resulting from the computation. It should be emphasized that a variable -- and only one variable -- can appear to the left of the equal sign in an algebraic statement in FORTRAN. The following has no meaning in FORTRAN:

AHC + 10.00 = 50.00

The computer reads statements (instructions) in a program from the top down unless specifically instructed to do otherwise. Furthermore, a succeeding algebraic statement can supercede a prior one. For example, the variable AHC is given two different values in the following three statements: AHC - 40.00, AOC = 25.00, and AHC = 60.00. The computer on encountering the first of the above statements stores $40.00 for the variable AHC, but by the time the last statement has been executed AHC has the value of 60.00, the 40.00 having been replaced by 60.00.

So far we have used only constants to the right of the equal sign in our algebraic statements. In most programs variables also appear on the right side of the equal sign. An appropriate FORTRAN algebraic statement might also be:

TAC = AHC + AOC.

Upon encountering the above algebraic statement the computer would add together the values currently stored for the variables AHC and AOC, and would store the value of the sum for the variable TAC. (By the time a statement such as that above is encountered, the variables AHC and AOC should have been given the desired values.)

Flexibility is permissible in writing variable names; however, there are a few rules, and several of the most important are:

1. Variables can consist of no more than six letters or six letters and numbers;
2. Every variable must begin with a letter;
3. All letters must be capitals (there are no lower case letters in FORTRAN);
4. It is unwise to end a variable with the letter F

OPERATIONAL SYMBOLS

Some FORTRAN operational symbols are like ordinary mathematical symbols, but some are not. The customary mathematical symbols, their meaning, and the corresponding FORTRAN symbols appear below:

MATH Symbol	MEANING	FORTRAN Symbol
+	Add	+
-	Subtract	-
÷	Divide	/
x	Multiply	*
$()^2$	Square	**2
$\sqrt{}$	Square Root of	SQRTF ()

The compound interest formula, which gives the amount (S) to which an initial deposit (P) will grow in n years when compounded at R per cent annually, is

$$S = P\,(1 + r)^n.$$

Determining the amount to which $1,000 invested at 4 per cent compounded annually would grow in six years is down by the following:

```
P = 1000.00
R = .04
N = 6
S = P * (1.0 + R) **N
```

THE HIERARCHY OF OPERATIONS

The mathematical operations performed in FORTRAN follow a strict hierarchy format. The computer first scans everything that appears to the right of the equal sign in an algebraic expression, then

> Everything which appears inside a set of parentheses is done first.
>
> Exponentiation is done next.
>
> Then multiplication and division are done.
>
> Finally, addition and subtraction are done.

The computer moves from left to right taking and executing each operation in the above sequence.

PRINT STATEMENT

There are several FORTRAN statements whose purposes are to input and output data. The PRINT statement provides for the printing out of one or more specified variables, the current values stored for these variables. Thus if the statement

```
PRINT ________, N, S
```

is inserted at the end of the compound interest program mentioned previously it would cause the value of N and the value of S to be printed, at the end of the program.

The blank in the PRINT statement indicates something omitted. Specifically, we have left out reference to what is termed a FORMAT statement. A FORMAT statement is used to specify the format in which we want the print-out to appear; i.e., whether the values of N

and S should appear side by side, one above the other, with some identifying terminology, or otherwise.

NUMBERED STATEMENTS AND THE "GO TO" STATEMENT

FORTRAN statements can be numbered. For many reasons it is useful to number many FORTRAN statements. These numbers appear to the left of the statement and are in fact a form of label.

It was noted earlier that, unless otherwise specified, statements in a FORTRAN program will be executed in top - to - bottom order. The "GO TO" statement is one of a variety of statements which can be used to change this order of execution. Every "GO TO" statement must have a number (a label) after the words "GO TO". Upon encountering a "GO TO", statement, the computer will proceed to the statement whose number appears after the words "GO TO", whether the statement whose number is given appears prior or subsequent to the "GO TO" statement. To understand the use of the "GO TO" statement examine the example problem shown.

With this program the computer is instructed to calculate the amount to which a deposit of $1,000, earning 4 per cent compounded, would grow in one year. The value of the deposit at the end of the year, S, is pointed out. This value is then reviewed as the initial deposit (P is set equal to S in the next - to - last statement), and the computer is sent back to calculate the amount to which the new deposit will grow at the end of the next year by the statement: GO TO 2.

NUMBERED STATEMENTS AND THE " GO TO" STATEMENT (CONT' D)

```
     P = 1000.00
     R = .04
     N = 1
2    S = P*(1.0 + R)**N
     PRINT _______, S
     P = S
     GO TO 2
```

As the above program is written, there is nothing to prevent the process being repeated indefitely with the exception of machine failure.

COUNTERS

The use of a "counter" would aid in the production of a table showing the results of each yearly computation. It would also indicate the number of times the "loop" or repetitive operation had been completed. In the following the variable "YEAR" serves as a counter, and every time statements number 2 through the GO TO statement are repeated the value OF THE VARIABLE YEAR IS increased by 1.

```
YEAR = 0.0
     P = 100000
     R = .04
     N = 1
2    S = P*(1.0 + R)**N
YEAR = YEAR + 1.0
PRINT ______, YEAR, S
     P = S
   GO TO 2
```

Again, the way the routine is written will cause it to loop endlessly, baring machine failure.

THE IF STATEMENT

The IF statement is a "conditional" GO TO statement. An illustrative example would be:

IF (YEAR - 10.0) 2, 4, 7.

When the above is encountered, the computer will evaluate the expression appearing inside the parentheses after the word IF in the same fashion that it evaluates the expression to the right of an equal sign in an algebraic statement. If the value of the result is negative, the computer will proceed immediately to statement number 2; if the value of the result is equal to 0, the computer will proceed immediately to statement number 4; if the value of the result is positive, the computer will proceed immediately to statement number 7. It does not matter whether statements 2, 4, and 7 appear before or after the IF statement.

In the compound interest problem we've been working with, an IF statement could be used in place of GO TO. This would limit the endless loop we were in by setting of, say, 10 years:

```
YEAR = 0.0
    P = 1000.00
    R = .04
    N = 1
  2 S = P* (1.0 + R)**N
    YEAR = YEAR + 1.0
   PRINT _______, YEAR, S
    P = S
    IF (YEAR - 10.0) 2, 4, 4
  4 CONTINUE
```

On the tenth repeated operation the value of the expression in parentheses becomes zero, and the flow of control is sent to statement number 4. In the above routine the expression is parentheses would never become greater than zero (positive), as a result only statements number 2 and 4 really needed. However, all IF statements must mention three statement numbers after the parentheses, and each of the numbers must refer to a statement in the program. To conform to this FORTRAN convention we simply write: IF (YEAR − 10.0) 2, 4, 4.

The CONTINUE statement seves only to provide a statement for the number 4. When the CONTINUE statement is encountered, the computer passes on to the statement which follows the CONTINUE statement.

```
   YEAR = 0.0
      P = 1000.00
      R = .04
      N = 1
    2 S = P*(1.0 + R)**N
      YEAR = YEAR + 1.0

  PRINT _____, YEAR, S
      P = S
  IF (YEAR - 10.0) 2, 4, 4.
    4 CONTINUE
```

STOP AND END STATEMENTS

The preceding routine is a complete operational routine except for the specification of the FORMAT for printing the answer (i.e. the blank area in the PRINT statement) and except for a terminal statement; that is a STOP and an END statement. (STOP statements can be used at

times without an END statement which must terminate all programs, must be preceded by a STOP statement.)

The complete routine would then have a STOP statement below CONTINUE, followed by the END statement below that.

THE DO STATEMENT

Like the "GO TO" and the "IF" statements, the "DO" statement influences the order of execution of statements in a program. The DO statement is also best explained by an example:

```
DO   4J = 5, 35, 3
```

In the above, the 4 following the word DO refers to a statement (statement number 4) which must follow the DO statement. Upon encountering the above DO statement the computer will give the variable J the value 5. It will then proceed down through statement number 4 as in any other program. Upon reaching statement 4 the computer will execute statement number 4 and then will return to the DO statement, whereupon it will increase the value stored for the variable J by 3 (making it 8), then it will proceed as before through statement number 4, returning again to the DO statement, increasing the value stored for the variable J by an additional 3 units (making it now 11), etc. This looping process will continue until J has grown to the point where to repeat the loop again would cause J to be greater than 35, whereupon the looping process is discontinued, and the computer proceeds to the statement which follows statement number 4 (the end of the DO loop) and continues through the remainder of the program.

The preceding is a simplified approach to the use of the DO loop. In actual use the process becomes more sophisticated. The following three comments should be remembered:

(1) If only two numbers follow the equal sign in the DO statement, the third number (the amount by which the value of the variable is to be increased each time the loop is repeated) is presumed to be 1.

(2) Variables, rather than numbers, can be used to the right of the equal sign in the DO statement, making it possible to cause the number of times the loop is to be repeated dependent upon prior calculations in the program.

(3) The variable to the left of the equal sign in the DO statement can, but need not necessarily, be employed in the program anywhere else except in the DO statement itself.

The following is an example of the "economy" in programming accomplished by use of the "DO" statement:

```
      P = 1000.00
      R = .04
      N = 1
     DO 3 J = 1, 10
      S = P*(1.0 + R)**N
      P = S
3     PRINT ______, J, S
      STOP
      END
```

THE FORMAT STATEMENT

In our previous use of the PRINT STATEMENT we employed a blank to indicate that something -- a statement number -- had been left out. A complete PRINT statement for one of our descriptive

examples might have been written:

PRINT 8, S.

The number 8 refers to a FORMAT statement whose number, or label, is 8. (ALL FORMAT statements must be numbered statements. A set of parentheses must follow the word FORMAT in a FORMAT statement. Inside the parentheses, the arrangement of the data to be printed out is specified. The FORMAT statement referred to in our PRINT statement above might be written:

8 FORMAT (1 HO, 10X, F8. 2)

The first expression in the parens, 1HO, specifies that prior to each print out (i. e. prior to printing the value of S, our deposit, at the end of each suceeding year) we want the printer to double-space. Each suceeding value for S, therefore, will be printed on a separate line, with a blank line in between. With the expression 1H1 in the first position in the parens, we could cause each value for S to be printed out on a separate page, (this is sometimes a desirable situation. Finally if we were content with single spacing we would employ the expression 1 H (bl) (i. e., we would leave a blank space after the H).

In the third expression, F8. 2, the F specifies that the value we expect to be printed out for S will have a fractional part; i. e., it will contain a decimal. Had we made provision throughout our program to deal only with integers, we would have used an I instead of a F specification.

The 8 in the 8. 2 specifies that 8 columns or spaces should be reserved for printing out the various values of S. The . 2 specifies that 2 spaces should be reserved for digits to the right of the decimal point. The space to be set aside for the print-out of data must be clearly specified in the FORMAT statement. One space must be reserved for each digit; one for the decimal point; and one for the sign of the number to be printed out (indicating whether it is a positive or negative quantity). If we expect the values of S to be 9999.00 or less, then 8 spaces would be adequate.

It is permissible to reserve more spaces than will be required, but care should be taken to reserve enough spaces so that no part of our answer gets truncated except that part which we want truncated. In the process of computing the values of S each time through our program, the computer will carry its calculations well beyond the two decimal places required in our program. (How far is dependent the word SIZE of the individual computer.) Each time the PRINT statement is encountered, the answer is rounded off to two significant places (since we specified two digits to the right of the decimal point in the FORMAT statement to which our PRINT statement refers), and will print out only the rounded answer, dropping the decimal digits that were carried during the computation.

The second expression in the parentheses, 10X, specifies that the eight spaces reserved for printing out the value of S should begin

10 spaces to the right of the left margin of the sheet on which the values of S are to be printed out. Thus we specify blank spaces with the use of X's.

When a great deal of data is to be printed out -- data of a number of different types -- it is frequently useful to provide identifying remarks for the printed-out data. This can be done with an H specification. To understand how the H specification might be used, recall the PRINT statement of our final compound interest program:

```
3 PRINT 9, J, S.
```

We will let 9 refer to a FORMAt statement which we will write as follows:

```
9 FORMAT (1 H   , 22 HAT END OF YEAR
   NUMBER, I4, 5X, 11 H BALANCE = $, F8 .2).
```

When our PRINT statement is executed, the computer will first cause to be printed out, beginning at the left margin, the following:

AT END OF YEAR NUMBER.

Twenty-two spaces were reserved for the group of words and spaces, as shown by the number 22 preceding the H, which precedes the first word of the group. The H indicates that the twenty-two characters and spaces which follow it should be printed out precisely as they appear inside the parens of the FORMAT statement.

In the next four spaces will be printed out the current value of the variable J, since four spaces are reserved for the value of J by the next expression in the FORMAT statement, I4. The next express-ion 5 X, will cause 5 spaces to be skipped. Next, the group for which

eleven spaces were probided in the FORMAT statement will be printed:

BALANCE = $.

Finally, in the next eight spaces the current value of the variable S will be printed, since eight spaces were reserved for the value of S by the expression F8. 2 in the FORMAT statement.

Our print-out for the first three times through the DO loop would therefore appear as follows:

AT END OF YEAR NUMBER 1 BALANCE = $1040.00

AT END OF YEAR NUMBER 2 BALANCE = $1081.60

AT END OF YEAR NUMBER 3 BALANCE = $1124.86

Notice that a plus sign would not get printed out for the values of S. Negative values are indicated by a minus sign on print-out, but positive values are simply given no sign at all, in conformance with algebraic custom. However, if a value in expected to be positive, space still must be reserved for the sign, whether or not it will be printed out.

In our PRINT and FORMAT statements above we have provided for printing out only two values, one for the current value of the variable J and one for the current value of the variable S. Note that the computer automatically associates the first variable mentioned in the PRINT statement with the space reserved by the second specification (the F specification) in the FORMAT statement, and so on.

It is possible to have a PRINT statement and a corresponding FORMAT statement where no numerical values of program variables are to be printed at all. Instead of the scheme used above for identi-

fying output data we might have used two sets of PRINT and FORMAT statements to create a two column matrix for output data, with a heading above each column. Thus, before the DO statement in the program shown we might have written:

```
PRINT 10
10 FORMAT (1 H0, 2X, 4H YEAR, 2X, 10 H BALANCE ($)
```

Then at the end of the DO loop, we might place the following two statements, with the first statement (the PRINT statement) inside the DO loop:

```
PRINT 11, J, S
11 FORMAT (1 H, 2X, I 4, 3X, F8. 2)
```

With this arrangement the print-out, by the third time through the DO loop, would appear as follows:

YEAR	BALANCE($)
1	1040.00
2	1081.60
3	1124.86

One has a rather high degree of freedom in arranging data print-outs since most computers are equipped to print up to 120 characters on one line. But note there must be no unaccounted - for spaces inside the parentheses of a FORMAT statement, and that each specification group inside the FORMAT parens is separated from the others by commas.

APPENDIX C

COBOL

The initial interest in the development of a commercially oriented compiler language came from the Department of Defense. Representatives of DOD and the manufacturers working with other interested groups met as a group called the Conference on Data Systems Languages (CODASYL). Their work in the development of COBOL appeared as a publication in April of 1960 with a revised edition in June of 1961. The initial compiler has been called COBOL-61. The maintenance of COBOL is under the control of a standing committee appointed by CODASYL. This committee has been responsible for an updated compiler called COBOL-65. The name COBOL was derived from COmmon Business Oriented Language.

STRUCTURE

The COBOL program consists of four parts called divisions. These are the IDENTIFICATION, ENVIRONMENT, DATA and PROCEDURE divisions. The essential logic of the program is contained in the PROCEDURE division. The data structure which the PROCEDURE division must work with is defined in the DATA division. These two contain the program. The IDENTIFICATION division is relatively small with a format that may be considered as fixed. This division exists as a means of self-documenting the program. Each entry or statement line on a coding sheet is a single card in the program deck. An example of the IDENTIFICATION division would be:

```
001001 IDENTIFICATION DIVISION.
001002 PROGRAM-ID.  PAYROLL.
001003 AUTHOR.  BRENDAN BEHAN.
001004 DATE-WRITTEN.  MARCH 17, 1963
001005 SECURITY.  QUESTIONABLE.
001006 INSTALLATION.  GUINESS NR.1.
001007 REMARKS.  This section could be brief or might
       continue for a period deemed sufficient to
       thoroughly explain the intent of the program.
```

The only entries that are required by COBOL are the first two, all others are at the option of the writer of the program.

The ENVIRONMENT division is supposed to contain all information needed to solve compatibility problems. This consists of a description of the computer on which the translation is to be done and a description of the machine on which the compiled object program is to be run. Any assignment of mnemonic labels for reference to peripheral equipments will be explained in this division. The source computer (the machine on which translation occurs) and the object computer are not necessarily the same equipment. It is quite conceivable though that in a small computer installation, they would be. An example of some

ENVIRONMENT statements:

```
001101    ENVIRONMENT DIVISION.
001102    CONFIGURATION SECTION.
001103    SOURCE COMPUTER.  BANSHEE-1.
001104    OBJECT COMPUTER.  GOLEM-1.
001105    6 TAPE UNITS
001106    SPECIAL-NAME.  TYPEWRITER IS TELEPRTR.
001107    INPUT-OUTPUT SECTION.
```

The INPUT-OUTPUT Section is concerned with programmer assigned names and the peripheral equipment they refer to. These assigned names will be used in the main program as a way of relating data files and the particular pieces of equipment on which they are read or written.

A word or two about the examples that have been given might be appropriate. The examples of IDENTIFICATION and ENVIRONMENT division statements are written as they would appear on a programmer's coding sheet. From the sheet they would be transcribed, each line to a separate punched card. In the case of the ENVIRONMENT DIVISION example, the card would be punched up with 001101 appearing in card columns 1 thru 6. The first three digits identify the coding page number and the next three the sequence of instructions. Column seven is set aside for use as a continuation indicator in cases where words must be split and carried from one line to the next. This is much the same as in normal usuage, where a word is split and carried to the next line, preceded by a dash. Card columns 8 thru 72 are used to write out the statement. In our example, ENVIRONMENT DIVISION would run from columns 8 thru 27.

PROCEDURE DIVISION

The first divisions have been devoted to describing the program, the equipment and the data files. The actual programming ofthe logical steps to be followed to solve the problem does not begin until we get to the PROCEDURE DIVISION. The output of this division will be responsible for the program in machine language which will direct the computer. The development of the program now duplicates the writing of a composition. The programmer's ideas must be expressed in meaningful English words, sentences and paragraphs. Each COBOL sentence will be ended with a period. The individual sentences will be formed around a verb or action word. The basis of the COBOL language is in its vocabulary which consists of a specific list of reserved words. The verbs in this list of reserved words fall into two general categories - processor and program verbs. Program verbs refer to the processing steps to be performed by the object program. The processor verbs are to direct the processor itself.

The following is a list of the COBOL verbs:

<u>PROGRAM VERBS</u>

INPUT/OUTPUT	OPEN READ WRITE CLOSE ACCEPT DISPLAY
<u>DATA MANIPULATION</u>	MOVE EXAMINE
<u>ARITHMETIC</u>	ADD SUBTRACT MULTIPLY DIVIDE COMPUTE
<u>SEQUENCE CONTROL</u>	GO TO ALTER PERFORM STOP

<u>PROCESSOR VERBS</u>

ENTER
EXIT
NOTE

The input/output verbs provide methods of placing data on an external or peripheral device (i.e. magnetic tape, magnetic disk units and other similar devices). They also provide the means for bringing data from these devices into the main computer operation.

The data manipulation verbs are used to transfer data from one area of storage to another and to do conversions or actual changes of items within the data structures.

The use of the various Arithmetic verbs is self evident in their various titles. COBOL provides additional options to allow for rounding, size error tests and data transfer with arithmetic operation.

Sequence control and procedure branching are synonomous terms indicating conditional and unconditional transfers of control and stops. Options to the statements are offered which allow transfers based on the results of certain programmed activities. For example:

```
IF PAYCARD EQUALS PAYROLL GO TO START.
```

The PROCESSOR VERBS are sometimes called compiler-directing statements. The ENTER statement permits communication between a COBOL object program and other COBOL subprograms or even other language subprograms. The EXIT verb is associated with PERFORM in sequence control to provide an end point to a procedure or paragraph. NOTE simply permits the programmer to write comments within the PROCEDURE DIVISION to be produced on the program listing. They do not effect the logic of the program.

The following is a short example of a PROCEDURE DIVISION:

```
003010   PROCEDURE DIVISION.
003020   PARAGRAPH-1.  OPEN INPUT IN-FILE,
003030       OUTPUT OUT-FILE
003040   PARAGRAPH-2.  READ IN-FILE RECORD;
003050       AT END GO TO PARAGRAPH-3.
003060       MOVE INVOICE-NUMBER IN IN-REC
003070       TO INVOICE-NUMBER IN OUT-REC.
003080       MOVE CUSTOMER IN IN-REC TO
003090       CUSTOMER IN OUT-REC. MOVE
003100       SALES IN IN-REC TO SALES
003110       IN OUT-REC. IF SALES IN
003120       IN-REC IS GREATER THAN
003130       100.00 MOVE 0.03 TO DISCOUNT
003140-      -PERCENT OTHERWISE MOVE 0.02
003150       TO DISCOUNT-PERCENT. MULTIPLY
```

```
003160       SALES IN IN-REC BY DISCOUNT-
003170-      -PERCENT GIVING DISCOUNT.
003180       SUBTRACT DISCOUNT FROM SALES
003190       IN IN-REC GIVING NET SALES.
003200       WRITE OUT-REC. GO TO
003210       PARAGRAPH-2
003220   PARAGRAPH-3. CLOSE IN-FILE, OUT-FILE.
```

DATA DIVISION

The DATA DIVISION is broken down into three sections- FILE SECTION, WORKING-STORAGE SECTION and the CONSTANT SECTION. They must appear in that order within the division if all are used. Any sections not needed may be omitted entirely. In general, the DATA DIVISION consists of a series of file descriptions, with a small amount of material (working areas and constants) which will not fit into a file structure. Our data structure is based on the FILE, which consists of RECORDS. The RECORD is sub-divided into ELEMENTARY ITEMS. If no sub-division is possible, a GROUP ITEM may then be an ELEMENTARY ITEM. The relationship of each of these is explained to the compiler in the DATA DIVISION by the use of level indicators. A typical entry follows:

```
002010 DATA DIVISION.
002020 FILE SECTION.
002030 FD  IN-FILE
002040     RECORDING MODE IS F
002050     BLOCK CONTAINS 10 RECORDS
002060     RECORD CONTAINS 38 CHARACTERS
002070     LABEL RECORDS ARE STANDARD
002080     DATA RECORD IS IN-REC.
002090 01  IN-REC.
002100     02 INVOICE-NUMBER PICTURE 9(4).
002110     02 CUSTOMER PICTURE X(28).
002120     02 SALES PICTURE 9999V99.
002130 FD  OUT-FILE
002140     RECORDING MODE IS F
002150     BLOCK CONTAINS 5 RECORDS
002160     LABEL RECORDS ARE STANDARD
002170     DATA RECORD IS OUT-REC.
002180 01  OUT-REC.
002190     02 INVOICE-NUMBER PICTURE 9(4).
002200     02 CUSTOMER PICTURE X(28).
002210     02 SALES PICTURE 9999V99.
002220     02 DISCOUNT-PERCENT PICTURE V99.
002230     02 DISCOUNT PICTURE 9(4)V99.
002240     02 NET-SALES PICTURE 9(4)V99.
```

In the example the "F" of FD appears in column 8 of our punched card. The 02 entries would begin at column 12. The FD stands for file description and provides the name of the file along with information required by the compiler. The name of the data record is necessary to show the relationship between the file and the sub-division (RECORD). A 01 level is then the RECORD, 02 is the GROUP ITEM and 03, the ELEMENTARY ITEM. The expression BLOCK describes the physical record as in the case of magnetic tape where logical records are blocked together to get as much data as possible on single tapes. The LOGICAL RECORD is then a sub-division of this BLOCK. A magnetic tape file may begin with

a title record or LABEL. The compiler is set up to recognize either the existence or absense of a label. This is then STANDARD or OMITTED in the FD entry. In the 02 descriptions PICTURE refers to the size of the area of this particular field.

The following characters are typical of those which may appear in a picture clause:

9 - This means the character in the picture will always be a digit (9(6) would mean a field of six digits, 9(9)-a field of nine digits).

V - A "V" indicates the position of an assumed decimal point. The point in COBOL is considered alphabetic. A compution must involve numeric values only.

A - This means the character in the picture (e.g. A(4) or AAAA) must always be a space or alphabetic.

X - When this appears in a picture (e.g. X(4) or XXXX) the picture can contain either alphabetic or numeric characters.

These are not all the characters used, there are others including "Z" for zero suppression, floating $, the asterisk * used as a space filler on output, the "S" for a sign (+ or -) and the "P" for decimal scaling.

SUMMARY

The information in this write-up is an attempt to give a general synopsis of COBOL. The actual programming of a problem in COBOL would require a great deal of specifics which have only been glossed over in this description. In general though what you have read is an introduction to COBOL as a programming language and covers a great deal of the material provided in the opening segments of a skill course. COBOL is an effective tool for any problem involving a situation commonly found in a commercial environment. It is used effectively in business in situations involving large volume input and summarized output or reports. In a recent survey of east coast commercial computer users, it was found that the majority of the 3000 companies queried used COBOL exclusively for their data processing.

GLOSSARY OF

MACHINE PROCESSING

TERMS

A

ACCUMULATOR.
1. The register and associated equipment in the arithmetic unit of the computer in which arithmetical and logical operations are performed.
2. A unit in a digital computer where numbers are totaled, i.e., accumulated. Often the accumulator stores one operand and upon receipt of any second operand, it forms and stores the result of performing the indicated operation on the first and second operands. Related to adder.

ADDRESS.
1. An identification, represented by a name, label, or number, for a register or location in storage.
2 Addresses are also a part of an instruction word along with commands, tags, and other symbols.
3. The part of an instruction which specifies an operand for the instruction.

ADDRESS, ABSOLUTE. An address which indicates the exact storage location where the referenced operand is to be found or stored, in the actual machine code address numbering system. Synonymous with (specific address) and related to code, absolute.

ADDRESS, ACTUAL. The address of a register or location in memory, in contrast with a relative or symbolic address. Same as address, absolute.

ADDRESS, BASE.
1. A number which appears as an address in a computer instruction but which serves as the base, index, or starting point for subsequent addresses to be modified. Synonymous with (presumptive address) and (reference address).
2. A number used in symbolic coding in conjunction with a relative address.

ADDRESS, EFFECTIVE.
1. A modified address.
2. The address actually considered to be used in a particular execution of a computer instruction.

ADDRESS, INDEXED. An address that is to be modified or has been modified by an index register or similar device. Synonymous with (variable address).

ADDRESS, RELATIVE. An address to which the base address must be added in order to find the machine address.

ADDRESS, SYMBOLIC. A label, alphabetic or alphameric, used to specify a storage location in the context of a particular program. Often, programs are first written using symbolic addresses which are translated into absolute addresses by an assembly program.

ALLOCATION, STORAGE. The process of reserving blocks of storage to specified blocks of information.

ALTERNATING TAPE DRIVES. See drives, alternating tape.

AREA, CONSTANT. A part of storage designated to store the invariable quantities required for processing.

AREA, INPUT. Same as block input, 1.

AREA, INSTRUCTION.

1. A part of storage allocated to receive and store the group of instructions to be executed.
2. The storage locations used to store the program.

ARGUMENT.

1. An independent variable; e.g., in looking up a quantity in a table, the number or any of the numbers which identifies the location of the desired value; or in a mathematical function, the variable for which a certain value is substituted to determine the value of the function.
2. An operand in an operation on one or more variables.

ASSEMBLE.

1. To integrate subroutines that are supplied, selected, or generated into the main routine by means of preset parameters, by adapting or changing relative and symbolic addresses to absolute form, or by placing them in storage.
2. To operate or perform the functions of an assembler.

ASSEMBLER. A computer program which operates on symbolic input data to produce machine instructions from such data by carrying out such functions as: translation of symbolic operation codes into computer operating instructions, assigning locations in storage for successive instructions, or computation of absolute addresses from symbolic addresses. An assembler generally translates input symbolic codes into machine instructions, item for item, and produces as output the same number of instructions or constants which were defined in the input symbolic codes. Synonymous with (assembly routine) and (assembly program), and related to compiler.

B

BACKSPACE. To move a magnetic tape one physical unit (e.g., record or file) in a reverse or backward direction to permit rereading or rewriting of the unit.

BIT.

1. An abbreviation of Binary digIT.
2. A single character in a binary number.
3. A physical representation of a binary digit; e.g., electrical pulses and magnetic spots on a tape, drum, or disk.

BIT, SIGN. A binary digit used as a sign digit.

BLANK.

1. A regimented place which contains no data but which may be used to store data; e.g., an empty location in a storage medium. Synonymous with (space).

BLOCK.

1. Data on magnetic tape between two consecutive inter-record gaps.
2. A set of one or more logical records handled collectively.
3. A group of consecutive machine words considered or transferred as a unit, particularly with reference to input and output.
4. To place data into such groups or blocks. Related to record, physical.
5. In a programming block diagram, a representation of a logical unit of computer programming.
6. A group of associated electrical circuits which may perform a specific function; it is often represented by a labeled rectangle in a block diagram showing the main functioning units of a piece of equipment.

BLOCK, INPUT.

1. A section of internal storage in a computer reserved for the receiving and processing of input information. Synonymous with (input area).
2. An input buffer.
3. A block of computer words considered as a unit and intended or destined to be transfered from an external source or storage medium to the internal storage of the computer.

BLOCK, OUTPUT.

1. A block of computer words considered as a unit and intended or destined to be transferred from an internal storage medium to an external destination.

2. A section of internal storage reserved for storing data which is to be transferred out of the computer. Synonymous with (output areas).

3. A block used as an output buffer.

BLOCKING. The combining of two or more records into one block.

BOOTSTRAP. A technique for loading the first few instructions of a routine into storage, then using these instructions to bring in the rest of the routine. This usually involves either the entering of a few instructions manually or the use of a special key on the console.

BRANCH. The selection of one of two or more possible paths in the flow of control, according to some criterion. The instructions which mechanize this concept are sometimes called "branch instructions"; however the terms "transfer of control" and "jump" are more widely used. Related to transfer, conditional.

BRANCH, CONDITIONAL. Same as transfer, conditional.

BRANCH, UNCONDITIONAL. Same as transfer, unconditional.

B-REGISTER.

1. Same as register, index.

2. A register used as an extension of the accumulator during multiply and divide processes.

BUFFER.

1. An internal portion of a data-processing system serving as intermediary storage between two storage or data-handling systems with different access times or formats; usually used to connect an input or output device with the main or internal highspeed storage. Clarified by storage, buffer, 4.

2. A logical OR circuit.

3. An isolating component designed to eliminate the reaction of a driven circuit on the circuits driving it; e.g., a buffer amplifier.

4. A diode.

BUG. A malfunction or a mistake in the design of a routine or a computer.

C

CALL, SUBROUTINE. In an object program, the set of in-line instructions that serves as a communications link between the main-line coding and an out-of-line subprogram and which is invoked by a subroutine call statement. Synonymous with (sequence, calling).

CHANNEL.

1. A path along which information, particularly a series of digits or characters, may flow.
2. One or more parallel tracks treated as a unit.
3. In a circulating storage, a channel is one recirculating path containing a fixed number of words stored serially by word.
4. A path for electrical communication.
5. A frequency or narrow band of frequencies of sufficient width for a single radio communication.

CLEAR. To erase the contents of a storage device by replacing the contents with blanks or zeros. Contrasted with hold and clarified by erase.

CLOCK.

1. A master timing device used to provide the basic sequencing pulses for the operation of a synchronous computer.
2. A register which automatically records the progress of real time or perhaps some approximation to it, and records the number of operations performed; the contents of the register are available to a computer program.

CODE, ABSOLUTE. A code using absolute addresses and absolute operation codes; i.e., a code which indicates the exact location where the referenced operand is to be found or stored. Synonymous with (one-level code) and (specific code) and rerelated to address, absolute.

CODE, INSTRUCTION. The list of symbols, names, and definitions of the instructions which are intelligible to a given computer or computing system. Related to repertoire, instruction.

COMMENT. An expression which explains or identifies a particular step in a routine but which has no effect on the operation of the computer in performing the instructions for the routine.

COMPARE. To examine the representation of a quantity to discover its relationship to zero; or to examine two quantities usually for the purposes of discovering identity or relative magnitude.

COMPILE. To produce a machine language routine from a routine written in source language by selecting appropriate subroutines from a subroutine library, as directed by the instructions or other symbols of the original routine, supplying the linkage which combines the subroutines into a workable routine, and translating the subroutines and linkage into machine language. The compiled routine is then ready to be loaded into storage and run--i.e., the compiler does not usually run the routine it produces.

COMPILER. A computer program more powerful than an assembler. In addition to its translating function, which is generally the same process as that used in an assembler, it is able to replace certain items of input with series of instructions, usually called subroutines. Thus, where an assembler translates item for item, and produces as output the same number of instructions or constants which were put into it, a compiler will do more than this. The program which results from compiling is a translated and expanded version of the original. Synonymous with (compiling routine) and related to assembler.

COMPUTER. A device capable of accepting information, applying prescribed processes to the information, and supplying the results of these processes. It usually consists of input and output devices, storage, arithmetic and logical unit, and a control unit.

COMPUTER, ANALOG. A computer which represents variables by physical analogies. Thus, any computer which solves problems by translating physical conditions such as flow, temperature, pressure, angular position, or voltage into related mechanical or electrical quantities and uses mechanical or electrical equivalent circuits as an analog for the physical phenomenon being investigated. In general, this type of computer uses an analog for each variable and produces analogs as output. Thus, an analog computer measures continuously, whereas a digital computer counts discretely. Related to machine, data processing.

COMPUTER, DIGITAL. A computer which processes information represented by combinations of discrete or discontinuous data as compared with an analog computer for continuous data. More specifically, it is a device for performing sequences of arithmetic and logical operations, not only on data but also on its own program. Still more specifically it is a stored

program digital computer capable of performing sequences of internally stored instruction, as opposed to calculators, such as card-programmed calculators, on which the sequence is impressed manually. Related to machine, data processing.

COMPUTER, GENERAL-PURPOSE. A computer designed to solve a large variety of problems; e.g., a stored-program computer which may be adapted to any of a very large class of applications.

COMPUTER, SPECIAL-PURPOSE. A computer designed to solve a specific class or narrow range of problems.

CONSTANTS. Quantities or messages which are present in a machine and available as data during a program run and which, usually, are not subject to change with time.

CONTROL.
1. The part of a digital computer or processor which determines the execution and interpretation of instructions in proper sequence, including the decoding of each instruction and the application of the proper signals to the arithmetic unit and other registers in accordance with the decoded information.
2. Frequently, it is one or more of the components in any mechanism responsible for interpreting and carrying out manually-initiated directions. Sometimes it is called manual control.
3. In some business applications, a mathematical check.
4. In programming, instructions which determine conditional jumps are often referred to as control instructions, and the time sequence of execution of instructions is called the flow of control.

CONVERSION, CODE.
1. In communications usage, that process by which a code of some predetermined bit structure, for example, 5, 7, 14, etc. bits per character interval is converted to a second code with more or less bits per character interval. No alphabetical significance is considered in this process. Clarified by code, 5; alphabet, 3; and interval, character. Contrasted with translation, alphabet.
2. In data processing usage, loosely, alphabet translation.

CONVERT.
1. To change numerical information from one number base to another.
2. To transfer information from one recorded medium to another.

COPY, HARD.

1. Printed machine processing output, as opposed to magnetic tape, paper tape, or other outputs.

2. In NSA, a written or printed copy of a message. Contrasted with copy, soft.

COUNTER. A device, register, or location in storage for storing numbers or number representations in a manner which permits these numbers to be increased or decreased by the value of another number, or to be changed or reset to zero or to an arbitrary value.

COUNTER, CONTROL. A device which records the storage location of the instruction word which is to be operated upon following the instruction word in current use. The control counter may select storage locations in sequence, thus obtaining the next instruction word from the subsequent storage location, unless a transfer or special instruction is encountered.

COUNTER, PROGRAM. Same as register, control.

CPU. Central Processing Unit. Same as frame, main, 1.

CYCLE-INDEX. The number of times a cycle has been executed, or the difference or the negative of the difference between the number that has been executed and the number of repetitions desired.

D

DATA.

1. A general term used to denote any or all facts, numbers, letters, and symbols, or facts that refer to or describe an object, idea, condition, situation, or other factors. It connotes basic elements of information which can be processed or produced by a computer. Sometimes data is considered to be expressible only in numerical form, but information is not so limited.

2. That which is given; the input to a process. Related to information.

DATA, RAW. Data which has not been processed. Such data may or may not be in machine-sensible form.

DATA-REDUCTION.

1. The process of transforming masses of raw, test, or experimentally-obtained data, usually gathered by automatic recording equipment, into useful, condensed, or simplified intelligence.

2. A manual or machine process transforming masses of raw data into useful form by removing redundant or irrelevant items. Related to editing, machine and to diarize.

DATA, TEST. A set of data developed specifically to test the adequacy of a computer run or system. The data may be actual data that has been taken from previous operations or artificial data created for this purpose.

DEBUG.

1. To locate and correct any errors in a computer program.

2. To detect and correct malfunctions in the computer itself. Related to routine, diagnostic.

DECISION. The computer operation of determining, at a certain time, if a certain relationship exists between words in storage or registers, and taking alternative courses of action. This is effected by conditional jumps or equivalent techniques. Use of this term has given rise to the misnomer "magic brain"--actually the process consists of making comparisons by use of arithmetic to determine the relationship of two terms (numeric, alphabetic, or a combination of both); e.g., equal, greater than, or less than.

DECISION, LOGICAL. The choice or ability to choose between alternatives. Basically this amounts to an ability to answer yes or no with respect to certain fundamental questions involving equality and relative magnitude; e.g., in an inventory application, it is necessary to determine whether or not there has been an issue of a given stock item.

DEGAUSS. To demagnetize magnetic tapes or other magnetic storage units in bulk, removing information contained on them by the use of devices especially designed for this purpose. Related to erase and clear; contrasted with record, 2.

DENSITY. In general, the number of characters, frames, or bits that can be stored in a storage medium per unit length or area.

DENSITY, CHARACTER. The number of characters that can be stored per unit of length; e.g., on some makes of magnetic tape drives, 200 or 556 bits can be stored serially, linearly, and axially to the inch.

DIAGRAM, BLOCK.

1. Any diagrammatic representation of the equipment or hardware components of a data processing system, showing their interrelationships and often their essential functions.

2. Such a diagrammatic representation in any field of endeavor. (Note: the term flowchart is now preferred for use in describing charts prepared in both systems analysis and programming work.) Related to flowchart; flowchart, program; and flowchart, system.

DOCUMENTATION. The group of techniques necessary for the orderly presentation, organization, and communication of recorded specialized knowledge, in order to maintain a complete record of reasons for changes in variables. Documentation is necessary not only to provide maximum usefulness but also to give an authoritative record for later reference.

DRIVES, ALTERNATING TAPE. In programming, two or more tape drives used to handle one multireel input or output file, permitting the switching of tapes without bringing the program to a halt. Also called ping-pong.

DUMP.

1. A transfer of selected items of information from storage for purposes of printing; used primarily as a diagnostic or debugging aid.

2. The material thus transferred or printed out.

3. A power dump; see dump, power.

DUMP, CHANGE, A print-out or output recording of the contents of all storage locations in which a change has been made since the previous dump.

DUMP, CORE. Same as dump, storage.

DUMP, MEMORY. Same as dump, storage.

DUMP, POWER. The removal of all power accidentally or intentionally.

DUMP, SNAPSHOT. A dynamic partial printout during computing, at breakpoints and checkpoints, of selected items in storage.

DUMP, STORAGE. A listing of the contents of a storage device or selected parts of it. Synonymous with (memory dump), (core dump) and (memory printout).

E

EDIT. To rearrange data or information. Editing may involve the deletion of unwanted data, the selection of pertinent data, the application of format techniques, the insertion of symbols such as page numbers and typewriter characters, the application of standard processes such as zero suppression, and the testing of data for reasonableness and proper range. Editing may sometimes by distinguished between input edit (rearrangement of source data) and output edit (preparation of table formats).

ENCODE.
1. To apply a code, frequently one consisting of binary numbers, to represent individual characters or groups of characters in a message. Inverse of decode.
2. To substitute letters, numbers, or characters for other numbers, letters, or characters, usually to intentionally hide the meaning of the message except to certain individuals who know the method employed.

END OF FILE. Termination or point of completion of a quantity of data. End-of-file marks are used to indicate this point. Synonymous with (EOF).

ENTRY.
1. A statement in a programming system. In general, each entry is written on one line of a coding form and punched on one card, although some systems permit a single entry to overflow several cards.
2. A member of a list.

ENTRY, KEYBOARD.
1. An element of information inserted manually, usually by a set of switches or marked punch levers called keys, into an automatic data processing system.
2. The capacity for achieving access to or entrance into an automatic data processing system as described in 1.

EQUIPMENT, OFF-LINE.
1. The peripheral equipment or devices not in direct communication with the central processing unit of a computer.
2. In communications or data processing, any equipments not electrically or physically interconnected, which are used in successive steps in a data-handling process. Related to on-line; off-line; processing, on-line; processing, off-line; and processing, in-line. Synonymous with (auxiliary equipment).

EQUIPMENT, ON-LINE.

1. Peripheral equipment or devices in direct communication with a central processing unit of a computer and under direct control of the computer.
2. Any equipments electrically interconnected, as in communications or data processing. Related to on-line; off-line; processing, on-line; processing, off-line; and processing, in-line.

EQUIPMENT, PERIPHERAL. The auxiliary machines which may be placed under the control of the central computer. Examples of this are card readers, card punches, magnetic-tape feeds and high-speed printers. Peripheral equipment may be used on-line or off-line depending upon computer design, job requirements, and economics. Clarified by equipment, automatic data-processing and by equipment, off-line.

EXECUTE. To interpret a machine instruction and perform the indicated operation(s) on the operand(s) specified.

F

FILE. An organized collection of information directed toward some purpose. The records in a file may or may not be sequenced according to a key contained in each record.

FILE, DETAIL. A file of information which is relatively transient. This is contrasted with a master file which contains relatively more permanent information; e.g., in the case of weekly payroll for hourly employees, the detail file will contain employee number, regular time, and overtime, the hours such employee has worked in a given week, and other information changing weekly. The master file will contain the employee's name, number, department, rate of pay, deduction specifications, and other information which regularly stays the same from week to week.

FILE, LOGICAL. A collection of one or more logical records. Clarified by record, logical and block.

FILE, TRANSACTION. A file containing current information used to update an existing file.

FLAG.

1. A bit of information attached to a character or word to indicate the boundary of a field.
2. An indicator used frequently to tell some later part of a program that some condition occurred earlier.

3. An indicator used to identify the members of several sets which are intermixed. Synonymous with (sentinel) and clarified by gap, 2.
4. A character or group of characters used to identify or enclose a unit of information, usually found in a constant position within records. Related to tag and symbol, machine control.

FLEXOWRITER. A trade name for an electric typewriter capable of reading and punching paper tape; sometimes used on-line with computers.

FLIP-FLOP. A bistable device that can be switched from one stable state to the other by successive excitations.

FLOWCHART. A graphic or diagrammatic representation for the definition, analysis, or solution of a problem, showing the sequence of operations, data, flow, and the like. Two principal types of flowcharts used in data processing are the system flowchart and the program flowchart. Related to flowchart, program; flowchart, system; flowchart, master; and flowchart, operational. Contrasted with diagram, block. Synonymous with (process chart) and (flow diagram).

FLOWCHART, MASTER.
1. A graphical portrayal of the broad processing steps that comprise a present or proposed system.
2. An overall, informative flow diagram, indicating the inputs, major processing steps, and outputs associated with the area under study.

FLOWCHART, PROGRAM.
1. A flowchart describing what takes place in a stored program. Symbols are used to represent specific operations and decisions, their sequence within the program, and the like.
2. A flowchart drawn in sufficient detail to permit a sponsor's problem requirement to be readily converted to machine coding and/or a program language. (Note: formerly often called a clock diagram). Clarified by flowchart. Contraster with flowchart, system, and synonymous with (flowchart, machine-logic).

FLOWCHART, SYSTEM. A flowchart describing the flow of data through all parts of a system. Symbols are used to describe this flow, and to show relationships among the data (and its medium), equipment, equipment operations, and manual operations. An entire program run or phase can be represented by a single processing symbol, with appropriate input/output symbols. Clarified by flowchart, and contrasted with flowchart program. Related to analysis, systems, and diagram, block.

FRAME, MAIN.

1. The central processor of the computer system. It contains the main storage, arithmetic unit, and special register groups. Synonymous with (CPU) and (central processing unit).

G

GAP, INTERRECORD. An interval of space or time, deliberately left between recording portions of data or records. Such spacing is used to prevent errors through loss of data or overwriting, and permits tape stop-start operations.

GAP, RECORD. An interval of space or time associated with a record to indicate or signal the end of the record.

GENERATOR, PROGRAM. A program which permits a computer to write other programs, automatically. Generators are of two types: (a) the character-controlled generator, which operates like a compiler in that it takes entries from a library tape, but unlike a simple compiler in that it examines control characters associated with each entry, and alters instructions found in the library according to the directions contained in the control characters. (b) The pure generator which is a program that writes another program. When associated with an assembler, a pure generator is usually a section of program which is called into storage by the assembler from a library tape and which then writes one or more entries in another program. Most assemblers are also compilers and generators. In this case the entire system is usually referred to as an assembly system. Related to language, problem-oriented.

GROUP-INDICATE.

1. To print indicative information from only the first record of a group of records, suppressing the same data until dissimilar indicative information is reached.
2. The data so printed. Clarified by information, indicative.

H

HARDWARE. The physical equipment or devices forming a computer and peripheral equipment. Contrasted with software.

HIGH-ORDER. Pertaining to the weight or significance assigned to the digits of a number; e.g., in the number 123456, the highest order digit is one; the lowest order digit is six. One may refer to the three high-order bits of a binary word, as another example. Clarified by order,3.

HIT.
1. A term used in retrieval or matching routines to represent an "equal," or comparison, found by a machine.
2. In file maintenance, the finding of a match between a detail record and a master record.

I

INDICATOR. A device which registers a condition, such as high or equal conditions resulting from a comparison, or plus or minus conditions resulting from a computation. A sequence of operations within a procedure may be varied according to the position of an indicator.

INDICATOR, END-OF-FILE. A device associated with each input and output unit that makes an end-of-file condition known to the routine and operator controlling the computer.

INITIALIZE. To set various counters, switches, and addresses to zero or other starting values at the beginning of a computer routine, or at prescribed points of a routine, as an aid to recovery and restart during a long computer run.

INPUT.
1. Information or data transferred or to be transferred from an external storage medium into the internal storage of the computer.
2. Pertaining to the routines which direct input as defined above, or the devices from which such information is available to the computer.
3. The device or collective set of devices necessary for input as defined in 1.

INPUT-OUTPUT (I/O). A general term for the equipment used to communicate with a computer and the data involved in the communication.

INQUIRY. A technique whereby the interrogation of the contents of a computer's storage may be initiated at a keyboard.

INSTRUCTION.

1. A set of characters which defines an operation, together with one or more addresses or no address, and which, as a unit, causes the computer to perform the operation on the indicated quantities. The term instruction is preferable to the terms command and order used in this sense; command is reserved for a specific portion of the instruction word, i.e., the part which specifies the operation which is to be performed, and order is reserved for the ordering of the characters, implying sequence, or the order of the interpolation or the order of the differential equation. Related to code, 1.

2. The operation or command to be executed by a computer, together with associated addresses, tages, indices, and the like.

INSTRUCTION, BRANCH. An instruction to a computer that enables the programmer to instruct the computer to choose between alternative subprograms, depending upon the conditions determined by the computer during the execution of the program. Synonymous with (transfer instruction).

INSTRUCTION, SYMBOLIC. An instruction in an assembly language directly translatable into a machine code.

I/O. Input/Output. See input-output.

J

K

K. A thousand; often used following numbers to signify quantities such as the size of the main storage of computers in terms of words, e.g., 8K or 16K.

L

LABEL. A set of symbols used to identify or describe an item, record, mesage, or file. Occasionally it may be the same as the address in storage.

LANGUAGE. A system for representing and communicating information or data between people or between people and machines. Such a system consists of a carefully defined set of characters and rules for combining them into larger units, such as words or expressions, and rules for word arrangement or usage to achieve specific meanings.

LENGTH, RECOD. The number of characters necessary to contain all the information in a record.

LENGTH, WORD. The number of characters in a machine word. In a given computer, the word length may be constant, or it may be variable.

LIBRARY, ROUTINE. A collection of standard, proven routines and subroutines by which problems and parts of problems may be solved.

LIBRARY, SUBROUTINE. A set of standard and proven subroutines which is kept on file for use at any time.

LIST, ASSEMBLY. A printed list which is the by-product of an assembly procedure. It lists, in logical instruction sequence, all details of a routine, showing the coded and symbolic notation next to the actual notation established by the assembly procedure. This listing is highly useful in the debugging of a routine.

LOAD.
1. To put data into a register or storage;
2. To put a magnetic tape onto a tape drive or to put cards into a card reader.

LOCATION. A storage position in the main internal storage which can store one computer word and which is usually identified by an address.

LOGIC.
1. The science dealing with the criteria or formal principles of reasoning and thought.
2. The systematic scheme which defines the interactions of signals in the design of an automatic data processing system.

LOOP.
1. A self-contained series of instructions in which the last instruction can modify itself and repeat the series until a terminal condition is reached. The productive instructions in the loop generally manipulate the operands, while bookkeeping instructions modify the productive instructions and keep count of the number of repetitions. A loop may contain any number of conditions for termination. The equivalent of a loop can be achieved by the technique of straight line coding, whereby the repetition of productive and bookkeeping operations is accomplished by explicitly writing the instructions for each repetition. Synonymous with cycle, 1.
2. Communications circuit between two private subscribers or between a subscriber and the local switching center.

LOOP, CLOSED. Pertaining to a system with feedback type of control, such that the output is used to modify the input.

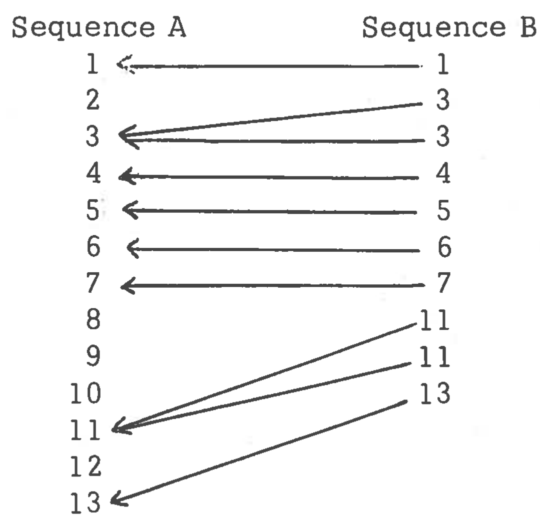
LOOP, OPEN. Pertaining to a control system in which there is no self-correcting action for misses of the desired operational condition, as there is in a closed loop system.

LOW-ORDER. Pertaining to the weight or significance assigned to the digits of a number; e.g., in the number 123456, the low order digit is six. One may refer to the three low-order bits of a binary word, as another example.

M

MARK, TAPE. The special character that is written on tape to signify the physical end of the recording on tape.

MATCH. A data processing operation similar to a merge, except that, instead of producing a sequence of items made up from the input, sequences are matched against each other on the basis of some key. The following is a schematic of a two-item match:



MEMORY PRINT-OUT. Same as dump, storage.

MERGE. To combine items into one sequenced file from two or more similarly sequenced files without changing the order of the items. Related to sort, four-tape, and sort, merge.

MESSAGE.
1. A group of words transported as a unit;
2. A transported item of information.

MNEMONIC. Pertaining to methods assisting human memory; thus, a mnemonic term, usually an abbreviation, that is easy to remember; e.g., MPY for multiply and ACC for accumulator.

MODIFY.
1. To alter a portion of an instruction so its interpretation and execution will be other than normal. The modification may permanently change the instruction or leave it unchanged and affect only the current execution. The most frequent modification is that of the effective address through use of index registers.
2. To alter a subroutine according to a defined parameter.

N

O

OFF-LINE. Descriptive of a system or equipment configuration in which the equipments involved in successive stages of data handling are not electrically interconnected. In machine processing usage, refers to input and/or output devices operating separately and not under the control of a central processing unit; in communications usage, refers to equipment involved in successive stages of message handling but not connected to the signal line. Contrasted with on-line, and related to processing, on-line; processing, in-line; and processing, off-line.

ON-LINE. Pertaining to a system or equipment configuration in which the equipments involved are electrically interconnected with each other. In machine processing usage, refers to input and/or output devices operating as a component of a computer system and under computer control; in communications usage, refers to equipment connected to the signal line. Contrasted with off-line; and related to processing, on-line; processing, in-line; and processing, off-line.

OPERAND.
1. A quantity or item of information entering into or arising in an operation described by an instruction. An operand may be an argument, a result, a parameter, or an indication of the location of the next instruction, as opposed to the operation code or symbol itself. It may even be the address portion of an instruction.

2. In ALPHA, a data-object operated upon by a procedural statement.

OPERATION, HOUSEKEEPING. General term for the operation which must be performed for a machine run-usually before actual processing begins. Examples of housekeeping operations are: establishing controlling marks, setting up auxiliary storage units, reading in the first record for processing, initializing, set-up verification operations, and file identification.

OUTPUT.
1. The information transferred from the internal storage of a computer to secondary or external storage or to any device outside of the computer.
2. The routines which direct 1.
3. The device or collective set of devices necessary for 1.
4. To transfer from internal storage onto external media.

P

PARAMETER.
1. A quantity in a subroutine, whose value specifies or partly specifies the process to be performed. It may be given different values when the subroutine is used in different main routines or in different parts of one main routine, but usually remains unchanged throughout any one such use. Related to parameter, program.

PATCH.
1. A section of coding inserted into a routine to correct a mistake or alter the routine. Sometimes, it is not inserted into the actual sequence of the routine being corrected, but placed somewhere else, with an exit to the patch and a return to the routine provided.
2. To insert corrected coding.

PLATEN. A backing, commonly cylindrical, against which printing mechanisms strike to produce an impression.

PRINTER, HIGH-SPEED (HSP). A printer which operates at a speed relatively compatible with the speed of computation and data processing so that it may operate on-line. At the present time a printer operating at a speed of 600 lines per minute, and 100 characters per line is considered high-speed.

PRINTER, LINE. A device capable of printing one line of characters across a page, i.e., 100 or more characters, simultaneously as continuous paper advances line by line in one direction past type bars or a type cylinder that contains all characters in all positions.

PROCESSING, IN-LINE. In a computer system, a method of data flow in which items as they occur are accumulated or temporarily stored for later processing but are kept in the original sequence and processed at intervals in small groups. Because in-line processing can be thought of as on-line processing done under relaxed time requirements, some authorities regard in-line processing as a kind of on-line processing and do not distinguish between the two. Refer to processing, on-line, and processing, off-line.

PROCESSING, OFF-LINE. A technique used in machine processing in which items to be processed are accumulated in a group, rearranged if necessary, and then introduced into a system at suitable intervals or when a batch of the proper size is available. Synonymous with processing, batch, and related to processing, on-line and processing, in-line.

PROCESSING, ON-LINE. A method of data flow in which information reflecting current activity is introduced into a data processing system as soon as it occurs and goes directly from input devices into the central processing unit with no intermediate sorting or classifying stages. An on-line processing system usually has an on-line output and may be a real-time processing system. Related to processing, off-line; processing, in-line; and processing, real-time.

PROGRAM.
1. The complete plan for the solution of a problem; more specifically, the complete sequence of machine instructions and routines necessary to solve a problem.
2. To plan the procedures for solving a problem. This may involve among other things the analysis of the problem, preparation of a flow diagram, preparing details, testing, and developing subroutines, allocation of storage locations, specification of input and output formats, and the incorporation of a computer run into a complete data processing system. Related to routine.

PROGRAM, OBJECT. The program which is the output of an automatic coding system. Often the object program is a machine language program ready for execution, but it may well be in an intermediate language. Synonymous with (target program) and (object routine), and contrasted with program, source.

PROGRAM SOURCE. A computer program written in a language designed for ease of human expression of a class of problems or procedures; e.g., symbolic or algebraic. A generator, assembler, translator, or compiler routine is used to perform the mechanics of translating the source program into an object program in machine language. Contrasted with program, object.

PROGRAMMING, SYMBOLIC. The use of arbitrary names or labels to represent addresses and machine operations in order to facilitate programming.

R

RECORD.

1. A group of related facts or fields of information treated as a unit.
2. A listing of information, usually in printed or printable form.
3. To put data into a storage device.

RECORD, ADDITION. A record that is merged into a file being updated.

RECORD, DELETION. A record that causes the elimination of some corresponding record from a file.

RECORD, FIXED-LENGTH. A record consisting of a definite number of characters. The restriction may be deliberate to simplify the speed processing or may be caused by the characteristics of the equipment used.

RECORD, HEADER. A term used to indicate a record having data common to subsequent line records within a specific control group. Contrasted with record, line.

RECORD, LOGICAL. A collection of data elements closely enough related to be customarily processed as a unit within a computer even though, in an external recording medium, the same stream of data may occupy any number of physical records or any portion of one physical record. Clarified by clock, pack, and unpack. Contrasted with record, physical.

RECORD, TRAILER.

1. A record which follows a group of records and contains pertinent data related to the group of records.
2. A continuation of another record, including appropriate information to link the continuation with a previous record or records. This connecting information may be machine- or manually-supplied indicative information, or may be part of the data of the previous record repeated for purposes of correlation. Related to information, indicative.

REGISTER. A hardware device used to store a certain number of bits or characters. A register is usually constructed of elements such as transistors or tubes and usually contains approximately one word of information. Common programming usage demands that a register have the ability to operate upon information and not merely store information; hardware usage does not make the distinction.

REGISTER, CONTROL. A register which holds the identification of the instruction word to be executed next in time sequence, following the current operation. The register is often a counter which is incremented to the address of the next sequential storage location, unless a transfer or other special instruction is specified by the program. Synonymous with (program counter) and contrasted with register, program, 1.

REGISTER, INDEX. A register which contains a quantity that may be used to modify addresses. Synonymous with B-register, 1; and (B-box)

REGISTER, PROGRAM. A register in which the current instruction of the program is stored. Synonymous with (instruction register) and contrasted with register, control.

REPERTOIRE, INSTRUCTION.
1. The set of instructions which a computing or data processing system is capable of performing.
2. The set of instructions which an automatic coding system assembles.

RESTORE. To return an index register, a variable address, or some computer word to its initial or preselected value. Related to prestore.

ROUTINE.
1. A set of coded instructions arranged in proper sequence to direct the computer to perform a desired operation or sequence of operations.
2. A subdivision of a program consisting of two or more instructions that are fuctionally related; therefore, a program. Clarified by subroutine and related to program.

ROUTINE, CLOSED. A routine which is not inserted as a block of instructions within a main routine but is entered by basic linkage from the main routine.

ROUTINE, EXECUTIVE. A routine which controls loading and relocation of routines and in some cases makes use of instructions which are unknown to the general programmer. Effectively, an executive routine is part of the machine itself. Synonymous with (monitor routine), (supervisory routine), and (supervisory program).

ROUTINE, INPUT. A routine, sometimes stored permanently in a computer, to allow reading of programs and data into the machine.

ROUTINE, OPEN. A routine which can be inserted directly into a larger routine without a linkage or calling sequence.

S

SENTINEL. An indicator used to identify the members of several sets which are intermixed; also referred to as flag. Clarified by gap, 2.

SHIFT. To move the characters or bits of a unit of information columnwise right or left. For a number, this is equivalent to multiplying or dividing by a power of the base of notation. Related to shift, arithmetic and shift, cyclic.

SHIFT, ARITHMETIC. To multiply or divide a quantity by a power of the number base; e.g., if binary 1101, which represents decimal 13, is arithmetically shifted twice to the left, the result is 110100, which represents 52, which is also obtained by multiplying 13 x 2 twice; on the other hand, if the decimal 13 were to be shifted to the left twice, the result would be the same as multiplying by 10 twice, or 1300. Related to shift and shift, cyclic.

SHIFT, CYCLIC. A shift in which the digits dropped off at one end of a word are returned at the other end in a circular fashion; e.g., if a register holds eight digits, 23456789, the result of a cyclic shift two columns to the left would be to change the contents of the register to 45678923. Synonymous with (circular shift), (end-around shift), (logical shift), (non-arithmetic shift), and (ring shift).

SHIFT, END-AROUND. Same as shift, cyclic.

SHIFT, LOGICAL. Same as shift, cyclic.

SIGN.

1. In arithmetic, a symbol which distinguishes negative quantities from positive ones.
2. An indication of whether a quantity is greater than zero or less than zero. The signs often are the marks + and - , respectively; but other arbitrarily selected symbols may be used as codes at a predetermined location in records, and interpreted by a person or machine.

SOFTWARE. A general term for a computer programming system, normally including items such as executive programs, programming and procedural languages with associated compilers and assembly systems, and associated programmer aids (e.g., loader programs, dump routines, I/O packages, etc.). In other words, software consists of those programming packages which a programmer finds available for program preparation and execution before he starts writing a source program; software does not include problem or object programs. Contrasted with hardware, and clarified by program, object and program, source.

SORT.

1. To order or arrange data in a specific sequence based on a defined segment of that data. Clarified by sequencing.
2. To choose from an input only records of certain classes and to output these records. Clarified by selecting.
3. To separate all records from a given input into two or more classes, placing the output on separate tapes or other media accordingly. Clarified by splitting.
4. The result of 1, 2, or 3. Related to sorting and merge.

SORT, MAJOR. A sort dependent upon the control field which changes least often; e.g., people's last names in a telephone directory. Clarified by sort, intermediate and sort, minor.

SORT, MERGE. A single sequence of items in order according to some rule; produced from two or more previously unordered sequences without changing the items in size, structure, or total number. Although more than one pass may be required for a complete sort, items are selected during each pass on the basis of the entire key.

SORT, MINOR. A sort dependent upon the control field which changes most often; e.g., people's middle initials in a telephone directory. Clarified by sort, intermediate and sort, major.

SORTING. A general term encompassing the ideas of sequencing, selecting, and splitting. Clarified by sequencing, selecting, and splitting; related to sort and merge.

SPONSOR. A person or organization using the facilities of a data processing service or organization; also called a "user."

STATEMENT. In computer programming, a meaningful expression or generalized instruction in a source language.

STATEMENT, DEBUG. In a programming language, a statement whose purpose is to invoke a debugging aid routine during the testing of an object program.

STATION INQUIRY. The remote terminal device from which an inquiry into computing or data processing equipment is made.

STEP, PROGRAM. A phase of one instruction or command in a sequence of instructions, i.e., a single operation.

STORAGE.
1. The term preferred to "memory."
2. Pertaining to a device in which data can be stored and from which it can be obtained at a later time. The means of storing data may be chemical, electrical, or mechanical.
3. A device consisting of electronic, electrostatic, or electrical hardware or other elements into which data may be entered, and from which data may be obtained as desired.
4. The erasable storage in any given computer.
5. The act of storing. Synonymous with (memory).

STORAGE, AUXILIARY. Storage with the primary function of augmenting the capacity of internal storage for handling data and instructions. Data from auxiliary storage must normally be transferred to internal storage to be operated upon. Examples of auxiliary storage are units characterized by medium speed access such as magnetic drums and magnetic disks. Synonymous with (secondary storage).

STORAGE, BUFFER.
1. A synchronizing element between two different forms of storage, usually between internal and external. For example, an input device in which information is assembled from external or secondary storage and stored ready for transfer to internal storage and held for transfer to secondary or external storage. Computation continues while transfers between buffer storage and secondary or internal storage or vice versa take place.
2. Any device which stores information temporarily during data transfers. Clarified by buffer.

STORAGE, EXTERNAL. The storage data on a device that is not an integral part of a computer main frame, but is in a form prescribed for use by the computer. The storage may be under control of the computer, but data must be transferred to internal storage before it can be operated upon.

Results of processing are ordinarily returned to external storage as operations are completed. Examples of external storage are on-line or off-line magnetic-tape units, magnetic disks, punch-card readers, and the like. Synononymous with (external memory) and contrasted with storage, internal . Clarified by on-line, off-line, and frame, main.

STORAGE, FLIP-FLOP. A memory system in which the primary element of storage is a flip-flop. See flip-flop.

STORAGE, INTERNAL. The storage of data on a device that is an integral part of a computer main frame. Internal storage is directly accessible to the arithmetic and control units of a computer and is customarily used for storage of instructions and for data currently being operated upon by a computer. Examples of internal storage are magnetic-core and thin-film memory. Synonymous with storage, main and (internal memory). Contrasted with storage, external.

STORAGE, MAGNETIC. A device or devices which utilize the magnetic properties of materials to store information.

STORAGE, MAGNETIC-CORE. A storage device in which binary data is represented by the direction of magnetization in each unit of an array of magnetic material, usually in the shape of toroidal rings but also in other forms such as wraps on bobbins. Synonymous with (core storage).

STORAGE, MAGNETIC-TAPE. A storage device in which data is stored in the form of magnetic spots on metal or coated plastic tape. Binary data is stored in the form of small magnetized spots arranged in column form across the width of the tape. A read-write head is usually associated with each row of magnetized spots so that one column can be read or written at a time as the tape traverses the head.

STORAGE, PROGRAM. A portion of the internal storage reserved for the storage of programs, routines, and subroutines. In many systems, protection devices are used to prevent inadvertent alteration of the contents of the program storage. Contrasted with storage, working.

STORAGE, WORKING. A portion of the internal storage reserved for the data upon which operations are being performed. Synonymous with (working space) and (temporary storage), and contrasted with storage, program.

STORE.

1. To transfer an element of information to a device from which the unaltered information can be obtained at a later time.

2. To retain data in a device from which it can be obtained at a later date.

3. In British usage, the section of a computer used for storing information; storage in U. S. usage.

SUBROUTINE.

1. The set of instructions necessary to direct a computer to carry out a well-defined mathematical or logical operation.

2. A subunit of a routine. A subroutine is often written in relative or symbolic coding even when the routine to which it belongs is not.

3. A routine which is arranged so that control may be transferred to it from a master routine and so that, at the conclusion of the subroutine, control reverts to the master routine. Such a subroutine is usually called a closed subroutine. Comment: The terms "routine" and "subroutine" are relative. A single routine may simultaneously be both a subroutine with respect to another routine and a master routine with respect to a third. Usually control is transferred to a single subroutine from more than one place in the master routine and the reason for using the subroutine is to avoid having to repeat the same sequence of intructions in different placed in the master routine. Clarified by routine.

SUBROUTINE, CLOSED. A subroutine not stored in the main path of the routine. Such a subroutine is entered by a jump operation and provision is made to return control to the main routine at the end of the operation. The instructions related to the entry and reentry function constitute a linkage. Synonymous with (linked subroutine).

SUBROUTINE, OPEN. A subroutine inserted directly into the linear operational sequence, not entered by a jump. Such a subroutine must be recopied at each point that it is needed in a routine. Synonymous with (subroutine, direct insert) and (subroutine, in-line).

SUPPRESSION, ZERO. The elimination of nonsignificant zeros to the left of significant digits, usually before printing.

SWITCH, SENSE. A switch on the computer console that can be set on or off manually to control execution of routines within an object program.

T

TABLE.

1. A collection of data in a form suitable for ready reference.
2. Frequently, data stored in sequenced machine locations or written in the form of an array of rows and columns for easy entry and in which an intersection of labeled rows and columns serves to locate a specific piece of data or information.

TABULATE. To print totals, differences, etc., derived from input data without listing all records. Distinguished from list, 1.

TAG. In a programming language, a label used to name statements or instructions of a program.

TAPE, CHADDED PAPER. A paper tape with the holes fully punched.

TAPE, CHADLESS PAPER. A paper tape with the holes partially punched so that each chad is left fastened by about a quarter of the circumference of the hole at the leading edge. This type of tape is useful when information is written or printed on the punched paper or when loose chads would present a problem. Chadless punched paper tape is normally sensed by mechanical fingers, for the presence of chad can interfere with reliable electrical or photoelectric reading of the tape. Related to chad.

TAPE, MAGNETIC. A tape or ribbon of any material impregnated or coated with magnetically sensitive material on which information may be placed in the form of magnetically polarized spots.

TAPE, PAPER. A strip of paper capable of storing or recording information. Storage may be in the form of punched holes, partially punched holes, carbonization or chemical change of impregnated material, or by imprinting. Some paper tapes, such as punched paper tapes, are capable of being read by an input device of a computer or by a transmitting device, either of which senses the pattern of holes which represent coded information.

TAPE, PROGRAM. A tape which contains the sequence of instructions required for solving a problem and which is read into a computer prior to running a program.

TAPE, PROGRAM JOB. The magnetic tape on which a chain of programs for a particular job is stored. Related to procedure, 2; job, machine processing; and tape, program.

TAPE, PUNCH. A tape, usually paper, upon which data may be stored in the form of punched holes. Hole locations are arranged in columns across the width of the tape. There are usually five to eight positions, or channels, per column, with data represented by a binary code system. All holes in a column are sensed simultaneously in a manner similar to that used for punch cards. Synonymous with (perforated tape) and (punched tape).

TIME, ACCESS.

1. The time it takes a computer to locate data or an instruction word in its storage section and to transfer it to its arithmetic unit where the required computations are performed.

2. The time it takes to transfer information which has been operated on from the arithmetic unit to the location in storage where the information is to be stored. Synonymous with (read time), and related to time, write; time, word, 2; and processing, real time.

TOTAL, INTERMEDIATE. The result when a summation is terminated by a change of control group that is neither the most nor the least significant.

TOTAL, MAJOR. The result when a summation is terminated by a change in the most significant control group.

TOTAL, MINOR. The result when a summation is terminated by a change in the least significant control group.

TRANSEMBLER. In NSA, a procedure-oriented software system designed to prepare program-job tapes automatically for a specific type of computer; each such system consists of a library of independent and generalized routines used in data handling operations (e.g., sorting or collating). These routines are automatically called for and sequenced by an executive program through the use of control cards. Related to procedure, 2; GFO; and system, operating.

TRANSFER.

1. The conveyance of control from one mode to another by means of instructions or signals.

2. The conveyance of data from one place to another.

3. An instruction for transfer.

4. To copy, exchange, read, record, store, transmit, transport, or write data.

5. An instruction which provides the ability to break the normal sequential flow of control. Synonymous with (jump), and (control transfer).

TRANSFER, CONDITONAL. An instruction which, if a specified condition or set of conditions is satisfied, causes the sequence of control to be switched to some specified location. If the condition is not satisfied, the instruction causes the computer to proceed in its normal sequence of control. A conditional transfer also includes the testing of the condition. Synonymous with (conditonal jump) and (conditional branch), and related to branch.

TRANSFER, UNCONDITIONAL. An instruction which switches the sequence of control to some specified location. Synonymous with (unconditional branch), (unconditonal jump), and (unconditional transfer of control).

TRANSLATOR.
1. A program whose input is a sequence of statements in some language and whose output is an equivalent sequence of statements in another language. Synonymous with (translating routine).
2. A translating device.

TRUNCATE. To drop digits of a number of terms of a series thus lessening precision; e.g., the number 3.14159265 is truncated to five figures in 3.1415. Related to round.

TYPE 9. In NSA, the English character set used in data processing printing; type 9 differs from commercial standard print sets in that some special symbols are different. Related to set, character.

U

UNIT, ARITHMETIC. The portion of the hardware of a computer in which arithmetic and logical operations are performed. The arithmetic unit generally consists of an accumulator, some special registers for the storage of operands and results, supplemented by shifting and sequencing circuitry for implementing multiplication, division, and other desired operations. Often called the "arithmetic and logic unit" and abbreviated "ALU."

UNIT, ASSEMBLY.
1. A device which performs the function of associating and joining several parts or piecing together a program.
2. A portion of a program which is capable of being assembled into a larger, whole program.

UNIT, COMPARING. An electromechanical device that determines the equality or inequality of two groups of times pulses or signals.

V

W

WORD, DATA. A word which may be primarily regarded as part of the information manipulated by a given program. A data word may be used to modify a program instruction or to be combined arithmetically with other data words.

WORD-LENGTH, FIXED. Pertaining to computers having the property that a machine word always contains the same number of characters, digits, or bits.

WORD-LENGTH, VARIABLE. Pertaining to computers having the property that a machine word may have a variable number of characters. Clarified by word and related to mark, word.

X

Y

Z